TU/e

A High-Level Petri Nets based modeling language for generation of BPEL4WS processes

Creation of a mapping between BPEL4WS and High-Level Petri Nets

Geert Zijlmans
Eindhoven University of Technology
Department of Mathematics & Computer Science
April 2006

Summary

This thesis describes the graduation assignment of Geert Zijlmans and its results.

In normal day life there are many processes around us, even ones that we do not notice directly because they appear normal to us (e.g. washing ourselves, living our life). In business there are also many processes which hold the business alive. Yet in business it is essential for a company to know which processes keep the business / company alive. The key item in this situation is information about the processes. Mostly the information is in the minds of the people that take part in a process, but when a person leaves the company also the piece of information about this process leaves the company, which might be crucial to the process itself.

This is where the modeling of processes starts. One of the important goals when creating a model of a process is information preservation about a process. If a company has its processes modeled, then this company can use these models for e.g. locating bottlenecks in the process, which were not visible before and better management of the company's resources.

Many modeling languages have been developed for the modeling of processes. One that forms a basis for most of the other modeling languages is Petri Nets, which has been discovered and defined by C.A. Petri in 1962 [PN, 1962]. Nowadays many extensions have been made to this modeling language of which High-level Petri Nets has become an international standard [HLPN, 2000]. The language is (extremely) suitable for defining (complex) processes. It also has a standard graphical modeling language (GML) defined for it, which speeds up the development of a model in this language.

Since the revolution of the Internet and other networks, more and more automated business processes use the capability of communicating with each other without the intervention of people. A fairly new language that has been developed by Microsoft, IBM, SAP, BEA Systems and Siebel Systems is the language Business Process Execution Language for Web Services (BPEL4WS) [BPEL4WS, 2003], which can be used for the modeling of formal specifications of business processes and business interaction protocols. BPEL4WS uses the Web Services Standards as a black-box for communication between business processes.

BPEL4WS is an XML-based modeling language for which no standard GML exists yet. That is why the TU/e issued a research of which the assignment runs as follows and this thesis is a result:

> *Create a graphical modeling language based upon Petri nets' semantics. It should be easy to transform models in this language to Business Process Execution Language for Web Services models.*

The approach to the creation of this GML has been the translation of the BPEL4WS elements to their equivalent in the GML of HLPN to which some extensions have been made. The GML of HLPN is used in this translation, because the GML of Classical Petri Nets did not have enough expressive power to translate the semantics and elements of BPEL4WS. The translation resulted in a set of handles (= a predefined set of arcs, places and transitions) which can be used to model a Web Services-related process. Since the handles are a one-to-one translation of their BPEL4WS equivalent one can generate a BPEL4WS process very simple from these handles.

A special tool, the PN-BPEL4WS Tool, has been developed for this assignment. One can use this tool to model a Web Services-related process using these handles. An extra feature of this tool is that an already existing BPEL4WS process can be loaded into this tool and be converted to its EPNML form. EPNML [EPNML, 2004] is an XML-based language that is used to define the GML of a Petri Nets process textually. In a tool that supports the EPNML standard, the EPNML form of the BPEL4WS process can be loaded to run simulations and / or perform analysis upon it.

Notational Conventions

In this document the following notational conventions are used:

- [Reference] : When a piece of text is between a “[” and a “]”, then this is a reference to an information source that is specified in the back of this thesis. An information source is specified by its title, author(s), institute(s) / company, publication date and if possible and URL to the location where the information source can be found.
- `Code` : When a piece of text has the font type “Courier New”, then the piece of text represents a piece of code or reference to a piece of code.
- *Term* : When a word has been written in italics, then the specific word is a term which definition can be found in the list of terms in Appendix A
- ***Term*** : When a word has been written in italics and in bold, then the specific word is a term which definition can be found in in [WfMC, 1999].

In the definition of code the following extra conventions apply:

- `<element>?` : The “?” indicates that the element is an optional element for the specific element that it has been defined in.
- `<element>*` : The “*” indicates that the element can exist zero or more times within the specific element that it has been defined in.
- `<element>+` : The “+” indicates that the element can exist one or more times within the specific element that it has been defined in.
- `attribute?` : The “?” indicates that the attribute is an optional attribute for a specific element that it has been defined in.
- `Alternative 1| Alternative 2` : The “|” indicates that the user has a choice between `alternative 1` and `alternative 2` when defining a specific attribute of an element.

Contents

SUMMARY 2

NOTATIONAL CONVENTIONS 3

CONTENTS 4

PREFACE 6

INTRODUCTION 7

CHAPTER 1: HIGH-LEVEL PETRI NETS & BPEL4WS 13

1.1. HIGH-LEVEL PETRI NETS 13
1.1.1. Classical Petri Nets 13
1.1.2. HLPN 18
1.1.3. Process Design with HLPN 21
1.1.4. PNML 22
1.1.5. Extensions to HLPN that are used in the GA 25
1.2. BPEL4WS 28
1.2.1. History 28
1.2.2. BPEL4WS 32
1.2.3. Process Design with BPEL4WS 39

CHAPTER 2: A GML FOR GENERATING BPEL4WS MODELS 41

2.1. INTRODUCTION 41
2.2. TRANSLATION MODEL 41
2.2.1. Creating the Translation Model 42
2.2.2. Example: Initial example of [BPEL4WS, 2003] 66
2.3. MAPPING MODEL 73
2.3.1. Creating the Mapping Model 73
2.3.2. Example: Initial example of [BPEL4WS, 2003] 74
2.4. CORRECTNESS OF THE TRANSLATION MODEL & MAPPING MODEL 79

CHAPTER 3: THE PN-BPEL4WS TOOL 81

3.1. INTRODUCTION 81
3.2. CONSTRUCTION 81
3.2.1. Handle, BPEL4WS, WSDL and EPNML classes 81
3.2.2. BPEL4WS and EPNML parsers 82
3.2.3. BPEL4WS and EPNML creators 82
3.2.4. Tool's engine 83
3.3. GUI ENVIRONMENT OF THE PN-BPEL4WS TOOL 84
3.3. RELATION WITH THE TM AND MM 87
3.4. SIMULATION OF THE PN-BPEL4WS TOOL'S RESULT(S) 87
3.4.1. EPNML and YASPER 87
3.4.2. BPEL4WS and Oracle BPEL Process Manager 88

CHAPTER 4: DOCUMENTATION SYSTEM **90**

4.1. INTRODUCTION 90
4.2. AUDIENCE 91
4.3. PURPOSE 91
4.4. CONSTRUCTION 92
4.3.1. Section: Title 93
4.3.2. Section: Toolbar 94
4.3.3. Section: Content 95
4.5. MAINTENANCE & EXPANSION 96
4.4.1. Adding new content to the DS 96
4.4.2. Updating content of the DS 97
4.4.3. Deleting content from the DS 97
4.6. QUICK TOUR 97
4.5.1. How-to install the DS 97
4.5.2. How-to open the DS 97
4.5.3. How-to navigate in the DS 97

CONCLUSION **97**

APPENDICES **97**

REFERENCES **97**

Preface

This thesis would not have existed, if I had found a job at the end of my study Computer Science at the Fontys Hogeschool Eindhoven. The job market in the year 2002 was really bad for “starters” in the Information Technology sector. My plan was that if I did not have a job before the first of September, then I would start a study at the Eindhoven University of Technology. Since no job really caught my attention, I decided to study Computer Science at the Eindhoven University of Technology.

Since I already had a background in computer science, I could start with the shortened version of the study Computer Science. In this shortened version one must attend mandatory courses which form the basis of the study, courses of a specialization area and courses that were in another area than the specialization area. My specialization area is Information Systems, which mainly consists of courses in process modeling and database techniques. My other area is Visualization, which mainly consists of courses about the displaying of (abstract) data graphically in such a way that it becomes understandable for a person.

I have always enjoyed my study at the Eindhoven University of Technology, mainly thanks to the teachers that were superior in comparison with the teachers at the Fontys Hogeschool Eindhoven. I can sincerely say that my knowledge of computer science has grown from the moment that I started this study.

After two years of study I could start with my graduation assignment of which this thesis is a result. That is why I want to thank some persons that have played a role in my graduation assignment and “career” at the Eindhoven University of Technology. These persons are in a random order:

- Natalia Sidorova, PhD : Supervisor and member of my graduation committee.
- Natalia Ioustinova, PhD : Member of my graduation committee.
- Prof. Paul De Bra, PhD : Member of my graduation committee.
- Reinier Post : Software developer for the tool YASPER.

And last but not least I want to thank my parents, who have always supported me.

January 2006,
Geert Zijlmans.

Introduction

Graduation Assignment

In the introduction of this thesis the graduation assignment (hereafter abbreviated as *GA*) is described. The graduation assignment has been issued by the Eindhoven University of Technology, Department of Mathematics & Computer Science[1] (hereafter abbreviated as *TU/e*). The GA is a research, which has been carried out at the TU/e by Geert Zijlmans, a graduate student at the TU/e. The mentor put forward by the TU/e is Natalia Sidorova, PhD, a member of the Architecture of Information Systems group (AIS) of the Department of Mathematics & Computer Science.

The GA description runs as follows:

> *Create a graphical modeling language based upon Petri nets' semantics. It should be easy to transform models in this language to Business Process Execution Language for Web Services (hereafter abbreviated as BPEL4WS* [BPEL4WS, 2003]*) models.*

In the next subsections of the introduction the following topics are discussed concerning the GA:

- Context in which the GA is performed.
- Subjects that play a role in the GA.
- Work that has to be done in the GA.
- Approach of the GA.

At the end of the introduction the layout of this thesis is described.

Note: This thesis uses the terminology for the description of business processes as defined in [WfMC, 1999]. A list of other terms that are also used in this thesis can be found in Appendix A.

Context of the GA

A logical question that arises is:

> *"Why issue this research?"*

Before I answer that question, we first have to look at the context of the research. This subsection describes this context and finally gives the answer to the question above.

In normal day life we do not directly notice them, but they are all around us: processes. A process may be a simple thing as shaving your beard, but may also be as complex as your own life. In business there are also a lot of processes, e.g. producing a product, cleaning the building, etc. In business it is very important to know what the involved ***business processes*** are.

Knowing your business processes starts with gathering information about them. Information is the key item in every business. Not having enough information about your business processes can have disastrous consequences for the business goal (producing products and / or offering services), e.g. producing products not knowing that there is no money to pay the employees, which is a mismanagement of resources; a strike by the employees is then most likely to happen.

Most of the information about business processes is in the minds of the people that take part in it. When these people leave the company, they also leave a big information gap about the business processes in which they took part. This is where modeling comes in. Modeling its business processes is the first step for a company to get an insight in what they are exactly doing and to preserve information about its business processes.

[1] http://w3.win.tue.nl/en/

A ***business process model*** (hereafter abbreviated as *BPM*) can be created using the information that is in the minds of the people that take part in the modeled business process.

Some important benefits for a company that has its business processes modeled are:

- Preservation of information about its business processes.
- Bottlenecks in the business processes may be solved (using the model).
- Possible improved efficiency of its business processes.
- Possible better management of resources (e.g. employees, money, etc.).

To model a business process one needs a *modeling language*. Until now several modeling languages have emerged such as Activity Diagrams of UML, Petri nets, XLANG, WSFL, BPEL4WS, eBPSS, and others. One of the most important modeling languages is Petri nets. This language is like the grandmother of most of the modeling languages. The language Petri nets has been discovered and defined by C.A. Petri in 1962 [PN, 1962] and several extensions have been made to this language after that (more on this subject in chapter 1). Besides that Petri nets is a basis for other modeling languages, it also has a strong academic background[2] (it is taught over several universities world wide, the TU/e is one of them).

Currently business processes become more and more automated. This also means that some of the business processes exchange information without the help of people. A new way that becomes standard is the exchange of information using Web Services. Web Services is a kind of black-box principle for exchanging information. The goal of the Web Services effort is to achieve interoperability between applications (created around / for business processes) by using Web Services Standards (see figure I.1 below).

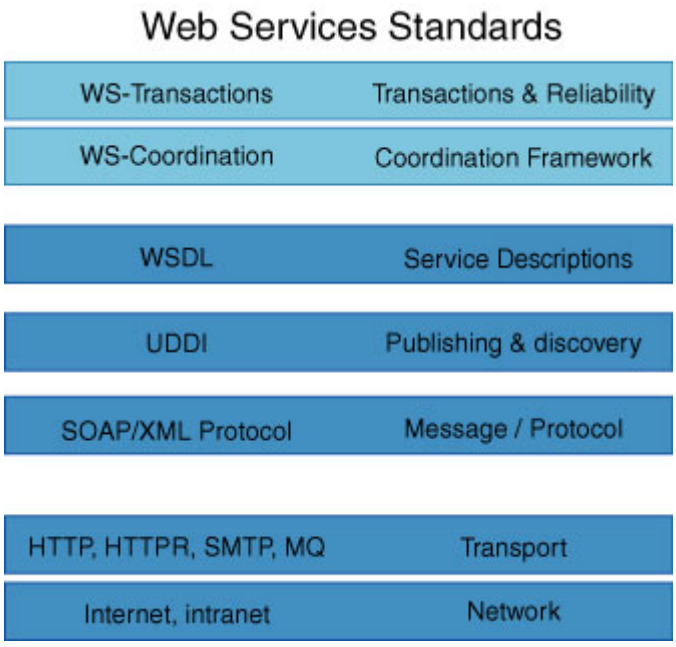


Figure I.1: Web Services Standards [IBM WS I, 2002].

[2] http://www.informatik.uni-hamburg.de/TGI/PetriNets/

BPEL4WS is a language that uses Web Services (hereafter abbreviated as *WS*) to enable processes to communicate with other processes (or people). Businesses can use BPEL4WS to model their formal specifications of business processes and business interaction protocols.
BPEL4WS is a merge of two "old" modeling languages, XLANG[3] (by Microsoft Corporation) and WSFL[4] (by IBM Corporation). The other parties that helped defining BPEL4WS are BEA Systems, SAP AG and Siebel Systems.

BPEL4WS is nothing more than another layer on top of the Web Services Standards (see figure I.1), which allows a business to use the Web Services Standards for the information exchange in its formal specification of business processes and business interaction protocols.
BPEL4WS is increasingly used to model processes like the online booking of trips. Yet there are not that many tools available for the modeling and testing of these models, because BPEL4WS is a fairly new modeling language (defined in 2002, version 1.1 in 2003).

Now that we have reached this point, I can give the answer to the question at the beginning of this subsection. Since Petri Nets is the basis for most of the modeling languages and since it has been there for a long time, many tools using Petri nets have been developed over the years, e.g. YASPER[5], CPN Tools, Design/cpn, etc.[6]. These tools can be used for modeling, testing and verification of business processes modeled in Petri nets. This is why the TU/e wanted a research on the creation of a Petri Nets-based *graphical modeling language* for BPEL4WS, which one can use to design models of business processes and business interaction protocols. If a business process or business interaction protocol designed with this Petri Nets-based graphical modeling language is tested and proved to be correct, than using the BPEL4WS representation of this model, one also knows that the BPEL4WS equivalent is correct. The translation between the Petri Nets-based graphical modeling language and the BPEL4WS representation should be simple.

Subjects of the GA

The GA includes roughly three subjects of which the first two are more or less the environment in which the GA takes place and the last one describes the work that has to be done.

Modeling languages for (business) processes:

Namely: Petri Nets and BPEL4WS. In the first chapter of this thesis these two modeling languages are described in short. For now it is enough to know that Petri Nets can be used to model ***workflows*** and BPEL4WS can be used to model business processes and business interaction protocols.

Graphical representation of a (business) process:

Defining and reading a BPM in its mathematical / textual form is a time consuming and a tedious job. This is why most of the modeling languages have a standard graphical modeling language (hereafter abbreviated as *GML*) on the basis of their semantics. A GML has two important advantages:

- It speeds up the development of a BPM.
- It improves the readability of a BPM.

Defining a GML for BPEL4WS based upon Petri nets' semantics:

Since BPEL4WS is a fairly new modeling language, it still has no standard GML. There have been studies to develop a (standard) GML for BPEL4WS (e.g. [WBERG, 2003]). This GA focuses on defining a GML based upon Petri nets' semantics from which it is easy to develop/generate a BPEL4WS model.

[3] XLANG is the language Microsoft uses on their BizTalk Servers. The specification of XLANG can be found on: http://www.godotnet.com/team/xml_wsspecs/xlang-c/default.htm
[4] The specification of Web Services Flow Language (WSFL) can be found on: http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
[5] YASPER homepage: http://www.yasper.org
[6] For a list of tools that use Petri nets see: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/complete_db.htm

Approach of the GA

In the previous subsections of this introduction the description, the context and the subjects of the GA are described. In this subsection the approach of the GA is described, which also resulted in the creation of this thesis.

The first step was to get acquainted with the environment in which the GA took place, which meant gathering information related to the GA. The first information I needed, was information about both modeling languages, Petri Nets and BPEL4WS.

My knowledge of the modeling language Petri Nets was already on a certain level that was sufficient for the GA to start with, since the language is part of a course at the TU/e (2M310 – Systems Engineering 1) that I attended. Nonetheless I reviewed all my documentation on Petri Nets (course material).

I had no knowledge of the modeling language BPEL4WS, which meant that I had to gather information about the modeling language BPEL4WS. This resulted in several documents about BPEL4WS of which its specification [BPEL4WS, 2003], the Web Services Description Language (hereafter abbreviated as *WSDL*) specification [WSDL, 2001] and tutorials by IBM[7] have been the most valuable to me to learn the modeling language BPEL4WS.

In my search for documents about the modeling language BPEL4WS, I also looked at some other modeling languages: UML Activity Diagrams [UML, 2005], Business Process Modeling Notation (BPMN) [BPMN, 2004], Web Services Flow Language (WSFL) [WSFL, 2001] and XLANG [XLANG, 2001]. I started a side-research (that was not part of the GA) to relate all these modeling languages (including Petri Nets and BPEL4WS) to each other, but this proved to be too much work and would be a research on its own, so I stopped it.

The next kind of information I needed was information about the GML of Petri Nets and existing GMLs for BPEL4WS. Petri Nets has a standard graphical notation and the only graphical notations that I could find for BPEL4WS was the one in the research by Daniela Weinberg [WBERG, 2003] and one in a tool by Oracle[8]. I was already acquainted with the graphical notation of Petri Nets, which meant that I had to take a look at the GML of the research and tool if they were usable for the GA. The GML of the research showed the importance of a place to store process instances and the tool showed the importance of nesting / hierarchical structuring / information hiding for the readability of the BPM.

The next search was for tools that are capable of using the (graphical) modeling languages (Petri Nets and / or BPEL4WS) to create BPMs. There are several tools available[9] for creating Petri Nets. One of them is YASPER. It has been developed at the TU/e in cooperation with Deloitte and uses the EPNML standard [EPNML, 2004]. EPNML is an application of the Petri Net Markup Language (hereafter abbreviated as *PNML*) [PNML, 2003], which is an XML-based markup language for the graphical notation of Petri Nets. Another tool that is available for BPEL4WS is BPEL4J by Alphaworks[10], which is a BPEL4WS editor.

This is where the work started. I tried and translated each element in BPEL4WS to its equivalent in the GML of Petri Nets. The idea was to find such a way of translating the elements that no information about the BPEL4WS process would be lost. Classical Petri Nets [PN, 1962] do not have enough expressive power to represent all the information that was in a BPEL4WS process, hence the choice for High-level Petri Nets [HLPN, 2000] was made. HLPN support the modeling of more complex structures, which means that more information could be preserved. The next step in the translation of the elements of BPEL4WS was to find patterns in the translated elements and to see if these patterns could be reused.

[7] The tutorial starts with [IBM BPEL 1, 2002] and continues from there on till [IBM BPEL 8, 2003].
[8] The Oracle BPEL Designer (http://otn.oracle.com/bpel) has its own GML for generation of BPEL4WS processes.
[9] http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html
[10] http://www.alphaworks.ibm.com/tech/bpws4j

These patterns were found and their use improved the readability of the translations drastically. Finally these translations have become part of the Translation Model (see Chapter 2 and Appendix E), which translates a BPEL4WS to its High-level Petri Nets equivalent.

The next thing that I wanted was to have the capability to validate and simulate a translated BPEL4WS process that was the result of the Translation Model. This meant I also had to translate each BPEL4WS element to a form that was understandable for the tools to work with. I chose EPNML because it is an application of the PNML standard and EPNML can be used by YASPER. When developing the translation of the graphical model to EPNML I noticed that I was losing information of the process instance and messages and because of this I introduced new extensions to EPNML, which can be done using the `<toolspecific>` element of PNML. Thanks to these new extensions no information was lost.

The next step was to retrieve standard patterns (= *handles*) from the Translation Model, which one can use to create a WS-related process model. This meant I had to group the (translated) elements of BPEL4WS according to their purpose in a *WS-related process*. After the grouping of the (translated) elements I created a Mapping Model (see Chapter 2 and Appendix F) which one can use to create a WS-related process model using these handles.

All together: the Translation Model, handles and Mapping Model describe the GML of this research.

Since I had to make extensions to the EPNML standard, I needed to build a tool which was capable of showing these extensions, and also was capable of applying the Translation Model and Mapping model and the use of the handles. This resulted in the PN-BPEL4WS Tool. Chapter 3 of this thesis gives an introduction to the use of the PN-BPEL4WS Tool.

As an extra I also created a Documentation System, which holds the most important information related to this GA.

Layout of the Thesis

This subsection gives a short introduction to the chapters of this thesis.

Chapter 1: High-level Petri Nets & BPEL4WS

The first chapter of this thesis describes the modeling languages High-level Petri Nets and BPEL4WS in short. The purpose of this chapter is to refresh the memory of the reader that is already acquainted with both modeling languages or in the other case to introduce the reader into the world of High-level Petri Nets and BPEL4WS. Both modeling languages are described in an informal way. This means that no heavy-duty mathematical definitions are given to describe both modeling languages. The formal specifications of both modeling languages can be found in the list of references (High-level Petri Nets [HLPN, 2000] and BPEL4WS [BPEL4WS, 2003]).

Chapter 2: High-level Petri Nets based GML for BPEL4WS

The second chapter describes the creation of a GML for BPEL4WS that is based upon the semantics of High-level Petri Nets. The first step in the creation of this GML has been the development of a Translation Model. The purpose of the Translation Model is to translate a WS-related process modeled with BPEL4WS to a WS-related process modeled with High-level Petri Nets. The second step in the creation of this GML has been the development of handles, which are retrieved from the Translation Model. The third step in the creation of this GML has been the development of a Mapping Model in which these handles are used to create a WS-related process. The handles together with the Translation Model and Mapping Model form the GML.

Chapter 3: PN-BPEL4WS Tool

Another result of the GA is an application that one can use to develop WS-related processes using the GML of this research, the PN-BPEL4WS Tool. The tool can only be used to model a WS-related process. This means that the validation and simulation of the model must occur in another tool. For this fact a model created in the PN-BPEL4WS Tool can be saved to its High-level Petri Nets form and to its BPEL4WS form, which can be used in a tool that is capable to validate and / or simulate the model. The purpose of the third chapter of this thesis is to give the reader a quick introduction to this tool and to show possible steps for validation / simulation of a model created in this tool.

Chapter 4: Documentation System

The Documentation System is a kind of reference manual of this GA. It contains the most important information about the GA: The specifications of BPEL4WS, WSDL and EPNML, the Translation Model and Mapping Model, and manuals of the PN-BPEL4WS Tool. The purpose of this chapter is to give the reader an impression of what the Documentation System is and what kind of information can be found in the Documentation System.

Conclusion

This part of the thesis is a concluding chapter in which several conclusions are made about the GA and its related topics. This part also describes several recommendations that are based upon the result of this GA. One might use these recommendations for further research in the area of the GA.

Appendices

In the appendices one can find a list of terms (except for the ones already defined in [WfMC, 1999]) and all abbreviations used in this thesis, BPEL4WS defined in Backus-Naur Form, a class model of BPEL4WS, the Translation Model, the Mapping Model and the initial example of the BPEL4WS specification translated using the Translation Model.

Chapter 1: High-level Petri Nets & BPEL4WS

1.1. High-level Petri Nets

If you are already acquainted with High-level Petri Nets (hereafter abbreviated as *HLPN*), then you can skip this subsection. This subsection describes the most important definitions and features of the modeling language HLPN in an informal way. It can be used by the experienced reader to refresh his/her knowledge about HLPN or for a novice reader to get an introduction into the world of HLPN. The formal specification of HLPN [HLPN, 2000] contains many mathematical definitions, which would only confuse the novice reader if these were mentioned here.

This subsection contains the following topics about HLPN:

- Classical Petri Nets : In order to describe what HLPN are, the reader first needs to know what Classical Petri Nets are.
- HLPN : Description of HLPN.
- Process Design with HLPN : Example of a simple WS-related process modeled with HLPN.
- PNML : Description of the Petri Net Markup Language
- Extensions : Description of HLPN extensions that are used in this GA.

The fields of application for Classical Petri Nets and HLPN include ([HLPN, 2000]):

- Requirements analysis;
- Development of specifications, designs and test suites;
- Descriptions of existing systems prior to re-engineering;
- Modeling business and software processes;
- Providing the semantics for concurrent languages;
- Simulation of systems to increase confidence;
- Formal analysis of the behavior of critical systems;
- Development of Petri Net Support Tools

In this GA the use of the modeling language HLPN focuses on the modeling of business processes.

1.1.1. Classical Petri Nets

The modeling language that has been defined by C.A. Petri in 1962 is now known as the modeling language Classical Petri Nets [PN, 1962]. Classical Petri Nets can be used to describe simple processes and systems.

When one looks at a Classical Petri Net model in its graphical form (see the small example in figure 1.1 below) one notices that it is a graph in which there are two different kinds of nodes: one is called places (elliptical) and the other one is called transitions (rectangular). These nodes are connected to each other via directed arcs. A developer can use these places, transitions and arcs to develop a BPM.

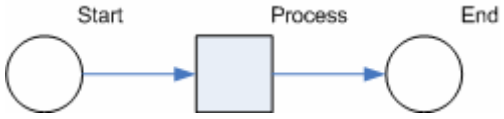


Figure 1.1: Example using the GML of Classical Petri Nets.

In this subsection the modeling language Classical Petri Nets is described from its graphical point-of-view. This means that first the use elements of the GML of Classical Petri Nets are described and then some important terminology that is used in the world of Petri Nets.

Transition:

A transition usually represents an (atomic) ***activity*** (an event, an operation, a transformation or transportation) that has to be performed in a business process. An *atomic activity* is an indivisible unit of work seen through the eyes of the person that is developing the BPM. After the execution of a transition the state of the net may change. A transition is represented by a rectangle in the GML of Classical Petri Nets. Also words can be associated with the rectangle to give the transition a (unique) name within the net, which identifies the transition (see figure 1.2).

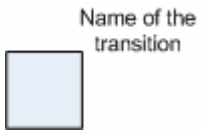


Figure 1.2: A possible graphical notation for a transition.

Place:

A place has the following purpose in a net:

- It is a storage place for tokens.
- It usually represents a medium, buffer, geographical location, state, phase or condition.

A place is represented by an ellipse in the GML of Classical Petri Nets. Also words can be associated with the ellipse to give the place a (unique) name within the net, which identifies the place (see figure 1.3).

Figure 1.3: A possible graphical notation for a place.

Seen from the perspective of a transition we differentiate between two types of places:

- Input place : A place p is called an input place of a transition t if and only if there is a directed arc from p to t. The set of input places for a transition t is denoted as •t. In the same way the set of transitions sharing place p as an input place is denoted as p•.
- Output place : A place p is called an output place of a transition t if and only if there is a directed arc from t to p. A set of output places for a transition t is denoted as t•. In the same way the set of transitions sharing place p as an output place is denoted as •p.

Directed arc:

A directed arc is represented by an arrow in the GML of Classical Petri Nets. Within a net there are two different types of arcs, seen from the perspective of a transition (see figure 1.4):

- Input arc : A directed arc that connects a place to a transition (Arc 1).
- Output arc : A directed arc that connects a transition to a place (Arc 2).

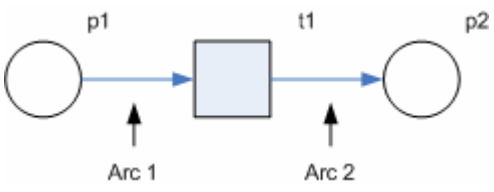


Figure 1.4: An input arc (Arc 1) and an output arc (Arc 2).

Token:

A token is represented by a black dot in the GML of Classical Petri Nets. Most developers draw tokens inside the place that they belong to (see figure 1.5).

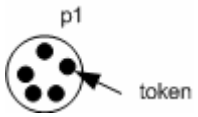


Figure 1.5: Five tokens in a place called 'p1'.

A token initially has no specific meaning in a Classical Petri Net, but it may get a meaning when the place where it is located represents one of the following inside a net:

- A resource : The place the token belongs to indicates that the token is a resource of some kind (e.g. the place's name is claim then the token inside can represent an instance of a claim).
- A control : The place the token belongs to is used in the execution of the process as a control mechanism (e.g. to limit the number of process instances that can run simultaneously).

Enabling of a transition:

A transition t may only fire if it is enabled. This occurs when there is at least one token in each place of •t. Enabling a transition t then means providing each place in •t with a token. Transitions are said to be concurrently enabled when they all have the possibility to fire.

Firing of a transition:

The following action within a net is called firing of a transition: A transition t removes one token from each place in •t and puts one token in each place of t•. In other words transition t consumes tokens from its input places and produces tokens for its output places.

State of a Petri Net model:

A state of a Petri Net model depends on its marking. A marking can be given by a vector of naturals of length |P|, where P is the set of places of the model. For example we have two places p1 and p2 and a transition t1 and two tokens in place p1. The marking of this situation in this little net is (2,0) where the first position in this vector represents the number of tokens in place p1 and the second position represents the number of tokens in place p2 (see figure 1.6). The firing of transition t1 causes one token to be consumed from place p1 and produced in place p2. The marking of the net in the new situation has become (1, 1) (see figure 1.7).

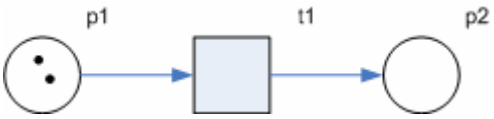


Figure 1.6: The marking of the Petri Net model is (2, 0).

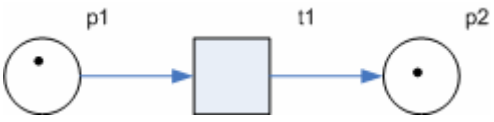


Figure 1.7: The marking of the Petri-net model after one firing of the transition t1 is (1, 1).

A ***state of a (business) process*** can be represented by a particular marking in a Petri Net of the (business) process. In our example we could use the marking (2, 0) as the state 'start' and the marking (0, 2) as the state 'end' of our little net.

Routing in a Petri Net

A routing in a Petri Net describes the order in which transitions in the model can be fired. In this GA there are three different kinds of routings, namely:

- Sequential routing : Sub-processes are fired in a sequential order;
- Parallel routing : Sub-processes are fired concurrently;
- Selective routing : There are several alternatives for process continuation;

These routing kinds can also be mixed, which means that a parallel routing may appear inside a sequential, etc.

Sequential routing:

In ***sequential routing*** the transitions belonging to the sequence are fired after each other. In figure 1.8 the transitions t1 and t2 are sequentially routed. This means that the token in place p1 first is consumed by transition t1, then transition t1 produces a token for place p2 and then transition t2 consumes the token in place p2 and produces a token for place p3.

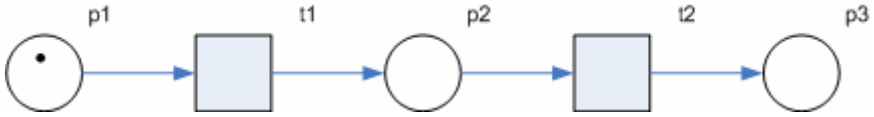


Figure 1.8: Sequential routing of transitions t1 and t2.

Parallel routing:

In ***parallel routing*** there are transitions that are concurrently enabled without conflict. In figure 1.9 there are four transitions. In this figure the transitions t2 and t3 are said to be concurrently enabled and thus can be executed in parallel. The transitions t1 and t4 respectively represent an AND-split and an AND-join which cause the part between these two transitions to be executed in parallel.

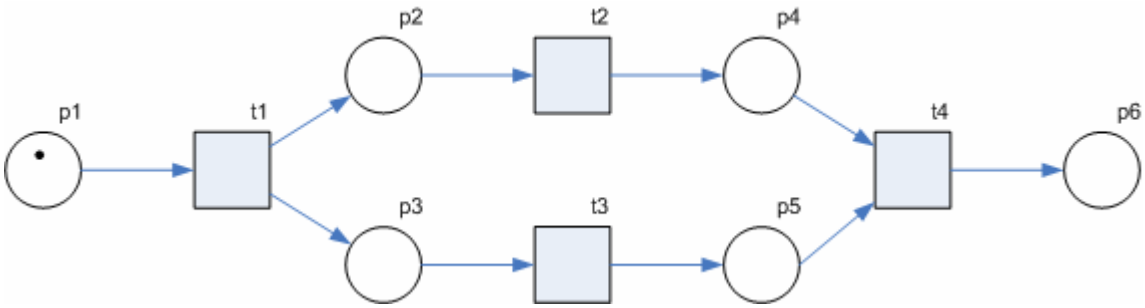


Figure 1.9: Parallel routing of transitions t2 and t3.

Selective routing:
In *selective routing* a choice has to be made between several alternative continuations (paths) of a process. In figure 1.10 two possible notations are shown. In the upper notation t3 and t4 are alternative paths inside a process. The upper notation explicitly defines which alternative to take depending on the firing of respectively transition t1 or t2 (XOR-split). If transition t1 fired, the transition t3 will fire, and after that transition t5 causes an XOR-join to be performed. If t2 fired, the transition t4 will fire and after that transition t6 causes an XOR-join to be performed. Once a path has been chosen, one cannot end up in another path defined in the same selective routing.
In the lower figure a non-deterministic choice is made between the two alternative transitions t3 and t4. The lower notation depicts an OR-split and an OR-join. Both notations can be combined resulting in either a combination of an XOR-split and an OR-join or a combination of an OR-split and an XOR-join.

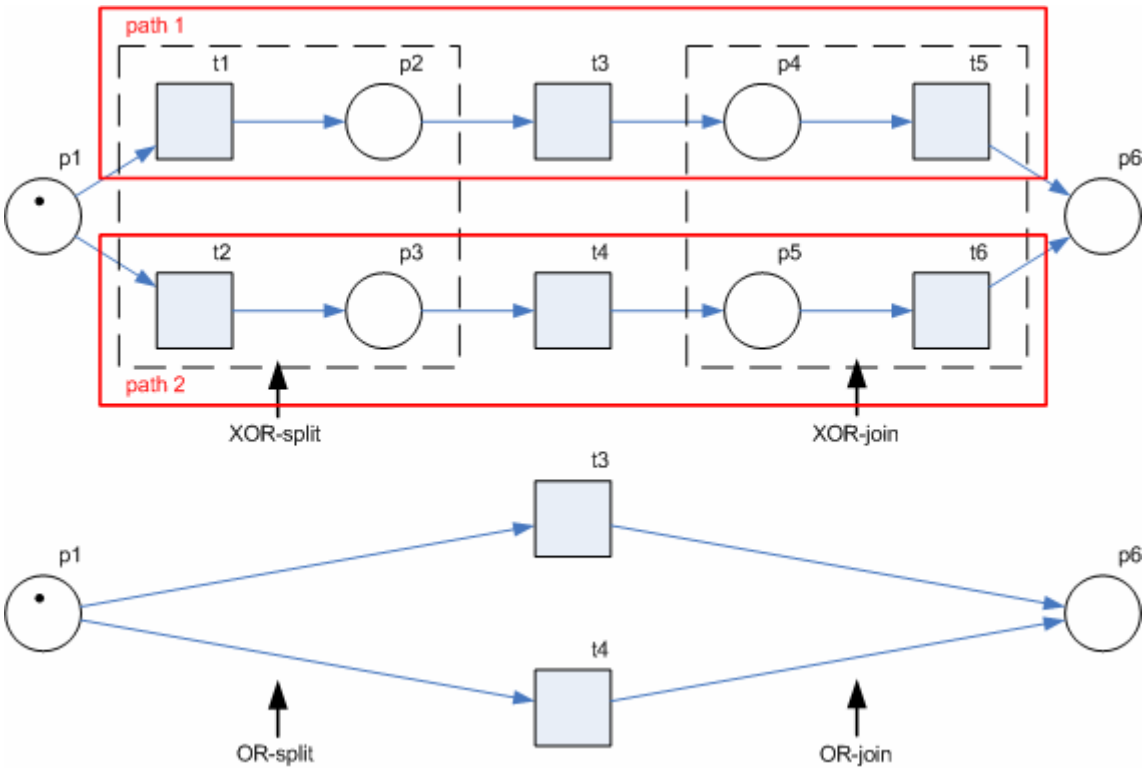


Figure 1.10: Selective routing of transitions t3 and t4.

1.1.2. HLPN

The modeling language High-level Petri Nets (HLPN) [HLPN, 2000] has become a standard in the year 2000 and can be used by developers to describe a wide range of complex (event) systems. In this GA it is used to describe business processes.

HLPN offers the developer of BPMs the same GML as used for Classical Petri Nets to which extensions have been made to be able to describe a complex system. This subsection describes the changes that have been made to each element in the GML of Classical Petri Nets, which forms the GML of HLPN. The example in [HLPN, 2000] on page 15 is used in this subsection, but has been adapted to make it understandable for a novice reader.

Transition:

The HLPN specification describes the extension of a transition with a ***transition condition***. A transition condition is a Boolean expression (e.g. Age >= 18) that is associated with a transition. A transition can only fire if the transition condition results into the value 'true'.
In the GML of HLPN a transition condition, if small, can be added to the representation of the transition, in the other case it must be mentioned in the description of the transition.
The small example (figure 1.11) below shows a transition that can only fire when a person is older than eighteen (the transition condition).

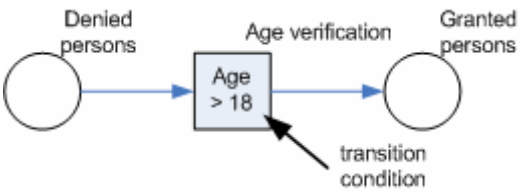


Figure 1.11: An example showing the transition condition "Age > 18" for a transition "Age verification".

Place:

Places in HLPN are extended with two new features described in [HLPN, 2000] as:

- Place Types : These are non-empty sets. One type is associated with each place.
- Place Marking : A collection of elements (data items) chosen from the place's type and associated with the place. Repetition of items is allowed. The items associated with places are called tokens.

The first feature describes the restriction of the kind of tokens that a place can harbor. The second feature describes the marking of a place at the start/initialization of the HLPN model as a collection of items of the type of the place.
Figure 1.12 below shows how a place of the previous example can be extended using these new features, where:

- P defines a set of the possible ages for a person;
- The marking 1'16 + 2'19 of a place means that there are three tokens present in that place upon initialization of the BPM, one of having the value 16 and two having the value 19;

Note: The meaning of a token in this place could be the following: it represents a person having the age that equals the value.

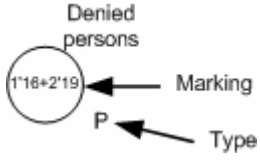


Figure 1.12: A place "Denied persons" having place type P and marking 1'16+2'19.

Token:

The second extension for a place implies that a token can be typed. This means that a token can represent a complex structure of data. This is a major advantage over Classical Petri Nets, where tokens simply represented "something". Now tokens can have an identity in the BPM and contain data, which can be used / altered by transitions in the process. A token always is related to a type definition of the place it belongs to. This means that prior to marking a place, the used type has to be defined.
A typed token is often referred to as a "colored" token, because often tokens of a particular identity are drawn by the developers as colored dots instead of the definition of the multi-set, like 1'16+2'19.
In figure 1.12a below the marking of the place "Denied persons" of figure 1.12 has been altered into three tokens of which a red token defines the value 16 and a green token defines the value 19.

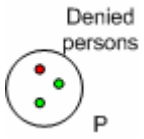


Figure 1.12a: A place "Denied persons" having place type P and marking 1'16+2'19, which are represented by colored dots.

Directed arc:

An arc in HLPN can be extended with an expression. According to [HLPN, 2000] the arc expression must result in a collection of items (tokens) taken from the type of the place it is connected to.
For a directed arc this means the following:

- If the arc is an input arc, then the arc describes a collection of items that can be used by the transition it is connected to (pre-condition for the transition).
- If the arc is an output arc, then the arc describes a collection of items that can be expected as an output of the transition it is connected to (post-condition for the transition).

In figure 1.13 below the arc is annotated with a variable Age that is of type P (as used in the previous example), which means that every item in the collection of P can be used as input / output.

Figure 1.13: A directed arc annotated with the variable Age.

Declarations:

Without the textual description of P and Age in the figures 1.11 to 1.13 one would not know what P and Age represent in the model. This is why HLPN [HLPN, 2000] uses declarations, which contain definitions of place types, typing of variables and function definitions that are used in the BPM.
In our small ongoing example the declaration of place types, typing of variables and function definitions can be defined as shown in figure 1.14 below, which is new element in the GML of Petri Nets.

```
P = {0..100}
>: ℤ X ℤ→Boolean arithmetic "more than"
Age: P
```

Figure 1.14: Declarations of place types, typing of variables and function definitions used in figure 1.11 to 1.13.

Figure 1.15 below shows the small example of figure 1.11 containing the extensions described by figures 1.11 to 1.14.

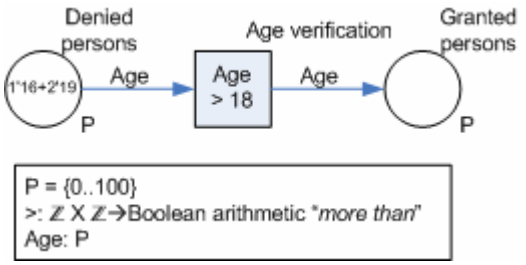


Figure 1.15: An example in which all extensions are used.

A logical description of this model could be that there are three denied persons, one having the age of sixteen and two having the age of nineteen. The model is bound to end in a state in which the marking of the place “Denied persons” is 1’16 and the marking of the place “Granted persons” is 2’19. This means that in the end state of the model there is one denied person having the age of sixteen and two granted persons having the age of nineteen.

Recommendation: The Z-Notation [ZNOT, 1998] can be used for the definitions of the declarations of a BPM.

1.1.3. Process Design with HLPN

A WS-related process is always initiated by another process or by people. This subsection describes a small example in which a "Request-Response" WS-related process is described and modeled (figure 1.16) using the GML of HLPN and the extensions mentioned in the previous subsection.

Example 1: "Age verification" WS-related process

On the Internet there are websites for which age verification is necessary. If a person is older than eighteen then the person may enter the website, else access will be denied to that person.
In this example a person arrives at a website for which age-verification is necessary. The following steps are taken by the person:

1. The person enters his / her age in a text field on the website;
2. The person presses a button on the website, which has an action coupled to it that sends the age entered in the text field to a web server to be checked by an "Age verification" WS-related process;
3. The person must wait until a response has been received from the web server;
4. If the response is positive, then the person may browse freely through the rest of the website, else the person closes the website.

The following steps are taken by the "Age verification" WS-related process:

1. The process waits until it receives a message containing the age of a person, which wishes to have access to the website;
2. The process then checks the content of the message. If the value of the content is above eighteen, then the response to the person is positive, else negative;
3. The process sends the response back.

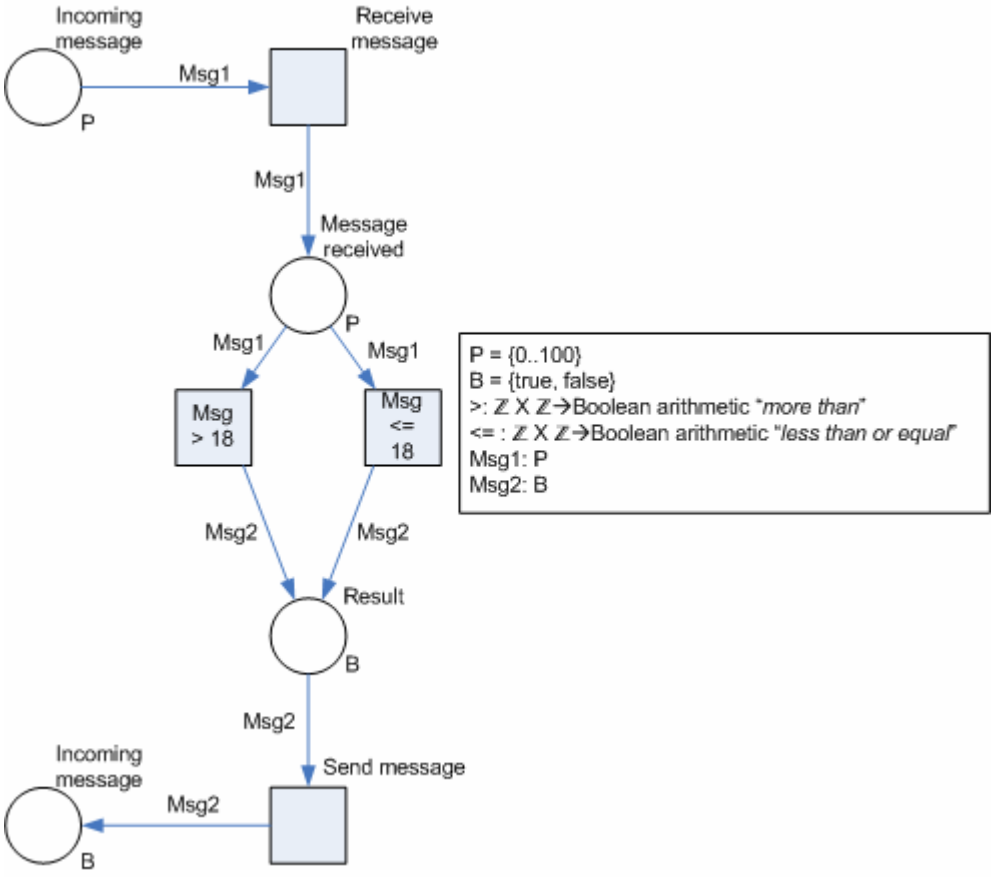


Figure 1.16: BPM of the "Age verification" WS-related process *of Example 1.*

1.1.4. PNML

PNML stands for the Petri Net Markup Language [PNML, 2003], which is an XML-based language that can be used to define the graphical representation of a Petri Net model.

In figure 1.17 below the PNML meta-model is shown in UML notation [UML, 2005]. The meta-model displays the classes that are used in the PNML standard. The exact description of these classes can be found in [PNML, 2003].

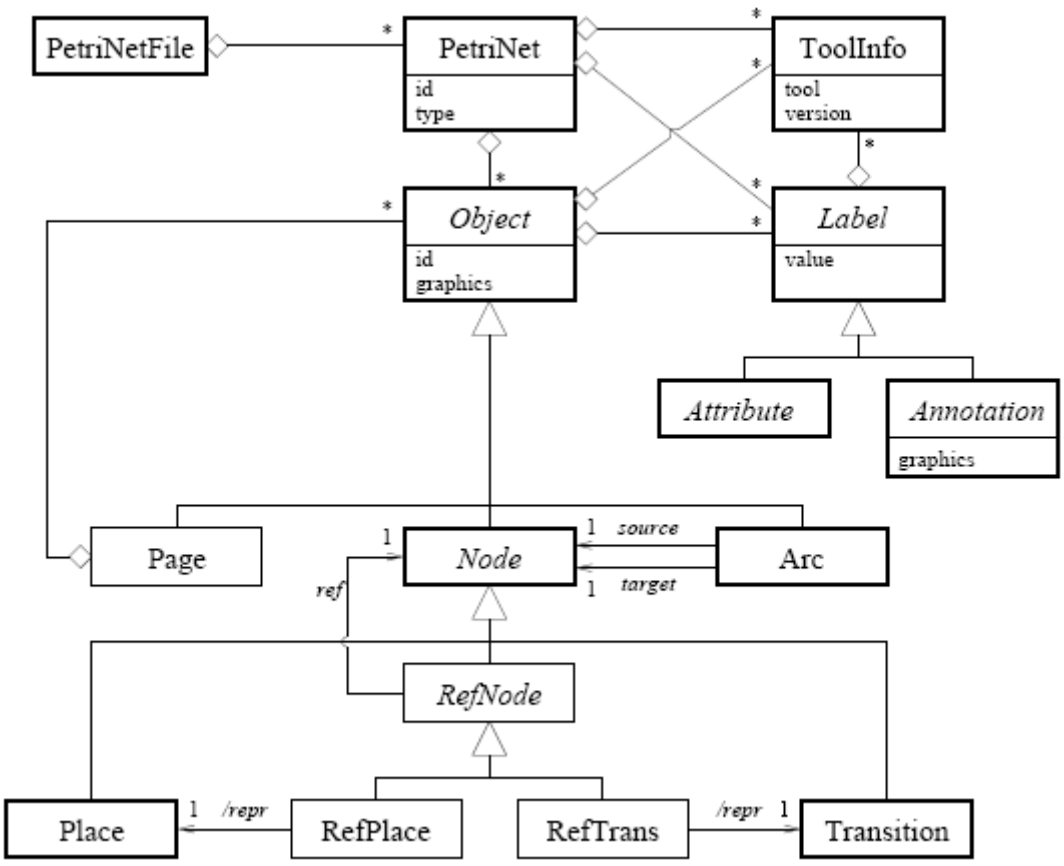


Figure 1.17: The PNML meta-model [PNML, 2003].

If one wants to define a Petri Net model using the PNML standard, one can use objects of these classes in the definition of the graphical representation of a Petri Net model, e.g. an object of the top class `PetriNetFile` describes a PNML file, which can contain zero or more objects of the class `PetriNet` that represent the Petri Net models contained in the PNML file, etc.

An object of a class in the meta-model can be translated to its PNML element using the translations provided by table 1.1 below, where the type `ID` is a unique identifier for the PNML-element in the whole PNML file and `IDRef` is a reference to a PNML element in the PNML file having the identifier indicated by this reference.

Class	XML Element	XML Attributes
PetriNetFile	**<pnml>**	
PetriNet	**<net>**	id: ID
		type: anyURI
Place	**<place>**	id: ID
Transition	**<transition>**	id: ID
Arc	**<arc>**	id: ID
		source: IDRef (Node)
		target: IDRef (Node)
Page	**<page>**	id: ID
RefPlace	**<referencePlace>**	id: ID
		ref: IDRef (Place or RefPlace)
RefTrans	**<referenceTransition>**	id: ID
		ref: IDRef (Transition or RefTrans)
ToolInfo	**<toolspecific>**	tool: string
		version: string
Graphics	**<graphics>**	

Table 1.1: Translation of the PNML meta-model into PNML elements [PNML, 2003].

Note: The class Graphics, which describes the graphical representation of an object or annotation, has been left out of the meta-model because it would clutter the model [PNML, 2003].

Primarily PNML can be used to define Classical Petri Net models, but one has the possibility to extend the language with new elements by defining a Petri Net Type Definition (PNTD) for the new elements. In the case of HLPN the PNTD has to define new elements for e.g. variables, transition conditions and arc inscriptions.

In this GA EPNML [EPNML, 2004] is used, which is an application of the PNML standard. In the table below the main EPNML elements with their attributes / sub-elements are displayed. Sub-elements are always optional.

Element	Attribute (A) / Sub-Element (SE)	Comment
<pnml>		Top level element of an EPNML file.
<net>	id (A)	Unique identifier for the net.
	type (A)	Default value: http://www.petriweb.org/specs/epnml11.
	<graphics> (SE)	Can be used for the graphical representation of the net.
	<name> (SE)	Can be used to name the net.
	<description> (SE)	Can be used for a description of the net.
<place>	id (A)	Unique identifier for the place.
	<graphics> (SE)	Can be used for the graphical representation of the place.
	<name> (SE)	Can be used to name the place.
	<description> (SE)	Can be used for a description of the place.
	<initialMarking> (SE)	Can be used to describe the initial marking of the place.
	<type> (SE)	Can be used to indicate if the place is a normal Petri net place (“channel”) or a place for data storage (“store”). Default value is “channel”.

Table 1.2: The main EPNML elements and their attributes / sub-elements.

Element	Attribute (A) / Sub-Element (SE)	Comment
<transition>	id (A)	Unique identifier for the transition.
	<graphics> (SE)	Can be used for the graphical representation of the transition.
	<description> (SE)	Can be used for a description of the transition.
	<transformation> (SE)	Can be used to specify token transformations performed by the transition.
	<type> (SE)	Can be used to indicate if the transition is an AND-transition, which has normal Petri Net semantics or an XOR-transition, which consumes non-deterministically a token from one of its input places and produces a token non-deterministically for one of its output places. Default value is “AND”.
<arc>	id (A)	Unique identifier for the arc.
	source (A)	Identifier that refers to a place or transition in the net
	target (A)	Identifier that refers to a place or transition in the net
	<graphics> (SE)	Can be used for the graphical representation of the arc.
	<name> (SE)	Can be used to name the arc.
	<description> (SE)	Can be used for a description of the arc.
	<inscription> (SE)	Can be used for defining arc expressions as used by HLPN [HLPN, 2000].
	<type> (SE)	Can be used to give the arc one of the following types: • inhibitor; if that place is a channel • C, R, U, and / or D; if that place is a store.
<page> (used for describing subnets)	id (A)	Unique identifier for the page.
	<graphics> (SE)	Can be used for the graphical representation of the page.
	<description> (SE)	Can be used for a description of the page.
	<type> (SE)	Can be used to indicate if the page is an AND-transition, which has normal Petri Net semantics or an XOR-transition, which consumes non-deterministically a token from one of its input places and produces a token non-deterministically for one of its output places. Default value is neither “AND” nor “XOR”.
<referencePlace>	id (A)	Unique identifier for the reference place.
	ref (A)	Identifier that refers to a place in the parent of the <page>-element the <referencePlace> belongs to.
	<graphics> (SE)	Can be used for the graphical representation of the reference place.
	<name> (SE)	Can be used to name the reference place.
	<description> (SE)	Can be used for a description of the reference place.

Table 1.2 (continued): EPNML elements and their attributes / sub-elements.

1.1.5. Extensions to HLPN that are used in the GA

In this GA we use some extensions that are not part of the HLPN specification [HLPN, 2000], but which may be supported in the future:

- Two kinds of place types: channel and store;
- New arc types related to the store
- Reset arcs;
- Augmentation of Petri Net models with time;
- Support for hierarchies in Petri Net Models;

Channel and store:

This extension is used in the EPNML specification [EPNML, 2004]. A channel is a regular Petri Net place, while a store serves as a data storage component[11]. A possible graphical representation of a store s1 of type A in a Petri Net is shown in figure 1.18 below.

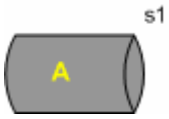


Figure 1.18: Graphical representation of a store s1.

The definition of a store in EPNML is shown in code 1.1 below.

```
<place id="placeID">
      <name><text>s1</text></name>
      <type><text>store</text></type>
</place>
```

Code 1.1: Definition of a reset-arc in EPNML [EPNML, 2004].

Note: The place's type A is lost in the translation to EPNML. In EPNML a store can only contain a datum.

New arc types related to the store:

The use of a store brings some new arc types that can be used in the model [EPNML, 2004]:

- C : The transition inserts a datum into the store.
- R : The transition reads[12] a datum from the store.
- U : The transition updates a datum in the store.
- D : The transition deletes a datum from the store.

These new arc types can also be combined, e.g. a transition can be connected to a store via an arc of the type "CRUD". Figure 1.19 below shows the graphical representation of these arcs.

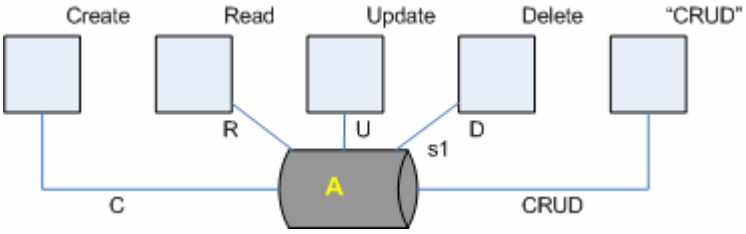


Figure 1.19: Graphical representation of arcs connected to a store s1.

[11] This concept is supported by the Petri Net tools: ExSpect (http://www.exspect.com) and YASPER (http://www.yasper.org).

[12] Actually the EPNML specification describes this type of arc as "the transition replaces a datum in the store". In this GA the description of the tool YASPER is used.

In code 1.2 below the EPNML definition of a typed arc is shown.

```
<arc id="arcID" source="sourceID" target="targetID">
      <type><text>type</text></type>
</arc>
```

Code 1.2: Definition of a typed arc in EPNML [EPNML, 2004].

Reset-arcs:

A reset-arc is a special arc type that is used in reset-nets [ResetPN, 1989]. The meaning of the reset-arc is different from a normal input arc: the place connected to a transition by a reset-arc loses all its tokens when the transition fires. The transition can fire even when there are no tokens inside the place. In this GA the reset-arc is used when terminating / halting a process in order to clear the model of tokens that have become obsolete (= tokens of the process that is terminated / halted). Figure 1.20 below shows the graphical representation of a reset-arc between a transition t1 and a place p1.

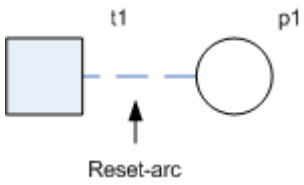


Figure 1.20: Graphical representation of a reset-arc.

The EPNML definition of a reset-arc (see code 1.3) is equal to the definition of a normal arc except for the piece of code that defines the type of the arc

```
<arc id="arcID" source="sourceID" target="targetID">
      <type><text>reset</text></type>
</arc>
```

Code 1.3: Definition of a reset-arc.

Time:

Petri Nets extended with time are called “Timed Petri Nets”, which already exist and are described in [TPN, 1998] by J. Wang.
In this GA the notion of time is added via a special place called “calendar”. A calendar is a store that always contains the current time in the model. In this GA time can be used in a transition to form a timer/alarm trigger and to provide each process instance with a value representing the time it was created.

The extension of transitions with time can be used for two possible transition conditions in this GA:

- A transition condition containing a duration expression, which means that the transition only can fire after a specified period of time.
- A transition condition containing a deadline expression, which means that the transition fires as soon as a specified moment in time has been passed.

A transition of one of the above described kinds has some kind of ‘wait’ functionality which iteratively reads the current time from the calendar until the transition condition results to ‘true’ and then fires. Figure 1.21 displays how a transition could be extended in this way, where A is a duration or deadline expression of the transition.

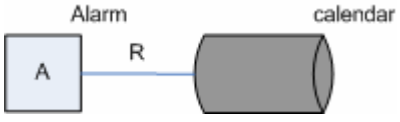


Figure 1.21: Graphical representation of an alarm-transition.

Hierarchies:

Petri Nets extended with hierarchies improve the readability of a BPM. A hierarchy in a Petri Net model describes an activity in which other activities may be nested. EPNML [EPNML, 2004] already supports this extension using the <page>-element of PNML. In figure 1.22 a possible graphical representation of a hierarchy h1 is shown, where:

- Places p1 and p2 are outside the hierarchy h1
- Place rp1 inside the hierarchy h1 refers to place p1 outside the hierarchy h1 and place rp2 inside the hierarchy h1 refers to place p2 outside the hierarchy h1.
- Transition t1 is performed inside hierarchy h1, which has place rp1 as its input place and rp2 as its output place.
- Zoom has the meaning of expanding a hierarchy to view the activities inside it.

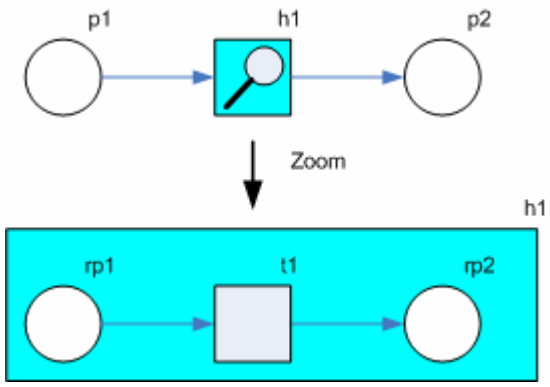


Figure 1.22: Graphical representation of a hierarchy.

As one might notice, one cannot conclude from the figure above that in hierarchy h1 the places rp1 and rp2 refer to places outside the hierarchy. That is why places like rp1 and rp2 mostly contain some kind of indicator that these places refer to places outside the hierarchy (the name of the reference place and the place referred to must be the same). The tool YASPER uses the following notation (see figure 1.23 below) for indicating that a place is a reference place of a certain kind. These indicators are also used in the figures of this thesis.

Figure 1.23: Indicators for reference places in hierarchies as used in YASPER.

Urgent transitions:

This is an extension to the definition of a normal transition as described earlier. An urgent transition described in [TPN, 1998] has the same features as a normal transition, except for the fact that the semantics for an urgent transition differs slightly from those of a normal transition: Once an urgent transition is enabled it must fire immediately without any delay.

When simulating a net that uses urgent transitions the following situations can occur:

- An urgent transition is concurrently enabled with a normal transition. In this case the urgent transition fires before the normal transition. After that, the normal transition may fire under the condition that there are no urgent transitions left that are enabled and that there are still enough tokens left for its firing.
- Two or more urgent transitions are concurrently enabled. In this case a non-deterministic choice is made between the firing of these urgent transitions.

1.2. BPEL4WS

If you are already acquainted with Business Process Execution Language for Web Services (BPEL4WS) [BPEL4WS, 2003], then you can skip this subsection. This subsection describes the most important definitions and features of the modeling language BPEL4WS in an informal way. It can be used by the experienced reader to refresh his/her knowledge about BPEL4WS or for a novice reader to get an introduction into the world of BPEL4WS.

This subsection contains the following topics about BPEL4WS:

- History : A little bit about the history of BPEL4WS.
- BPEL4WS : Description of BPEL4WS.
- Process Design with BPEL4WS : Example of a simple WS-related process modeled with BPEL4WS.

1.2.1. History

BPEL4WS is a merge of two competing modeling language standards that can be used to model a WS-related process:

- XLANG [XLANG, 2001] by Microsoft.
- Web Services Flow Language (WSFL) [WSFL, 2001] by IBM

Yet BPEL4WS has been developed by BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems as a joint effort to develop one leading standard that can be used for designing WS-related processes. This subsection describes its predecessors by indicating their most important features.

XLANG:

Microsoft's XLANG is a notation for specifying the message exchange behavior of a WS-related process.

XLANG is an extension for WSDL [WSDL, 2001]. An XLANG service description is a WSDL service description extended with elements using the namespace `xlang` (http://schemas.microsoft.com/biztalk/xlang/), which describe the behavior of the service as part of a business process.

XLANG has the following activity elements, called the *basic control processes* in XLANG, which can be used to model the message exchange behavior of the process:

- `<empty>` : This activity element represents the activity where "nothing is done".
- `<sequence>` : This activity element contains zero or more actions or basic control processes which are executed in sequential order.
- `<switch>` : This activity element supports conditional behavior. Each branch of this activity element contains a condition. If the condition evaluates to "true", then the specific branch is executed. If none of the conditional branches evaluates to "true", then a `default` branch is executed. If the `default` branch is not defined then an `<empty>` activity element is executed.
- `<while>` : This activity element provides a construct that iterates a basic control process until the given iteration condition evaluates to "false".
- `<all>` : This activity element contains zero or more actions or basic control processes which are executed concurrently.
- `<pick>` : This activity element is an event handler, which allows the process to wait for the occurrence of a particular event and then executes the basic control process defined for the specific event.

XLANG also provides means for exception handling, raising errors, duration and deadline expiration, compensation activities.
In code 1.4 below the example of the XLANG specification is given, which models the interaction behavior of a stock quote provider. In this example it is easy to see the behavior of the WS-related process it defines (XLANG part is defined in **bold**):

- First it performs the operation `AskLastTradePrice` of the port `pGetRequest`, where it waits for a request to arrive for retrieving the last trade price for a certain stock. The attribute `activation` informs if the action initializes the behavior for the WS-related process.
- Second it performs the operation `SendLastTradePrice` of the port `pSendResponse`, where it sends the last trade price of the requested stock to the requester.

```
<?xml version="1.0"?>
<definitions name="StockQuoteProvider"
             targetNamespace="http://example.com/stockquote/provider"
             xmlns:tns="http://example.com/stockquote/provider"
             xmlns:xlang="http://schemas.microsoft.com/biztalk/xlang/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

      <!-- WSDL type and message definitions used -->
      <!-- below have been omitted here for brevity -->
      <portType name="RequestReceivePortType">
            <documentation>
                  This port receives an asynchronous message requesting
                  the price of the given stock.
            </documentation>
            <operation name="AskLastTradePrice">
                  <input message="..."/>
            </operation>
      </portType>

      <portType name="ResponseSendPortType">
            <documentation>
                  This port sends an asynchronous message providing
                  the price of the given stock.
            </documentation>
            <operation name="SendLastTradePrice">
                  <output message="..."/>
            </operation>
      </portType>

      <binding name="RequestReceivePortBinding"
                  type="tns:RequestReceivePortType">
            <!-- details omitted -->
      </binding>

      <binding name="ResponseSendPortBinding"
                  type="tns:ResponseSendPortType">
            <!-- details omitted -->
      </binding>

      <service name="StockQuoteProviderService">
            <documentation>
                  The externally visible behavior of a simple
                  asynchronous stock quote provider service. A price
                  quote request arrives via E-mail. The request contains
                  a ticker symbol and the address to which we should send
                  the stock price. There is an invisible backend
                  implementation that obtains the quote and the service
                  then returns this to the appropriate address.
            </documentation>
            <port name="pGetRequest"
                  binding="tns:RequestReceivePortBinding">
```

```
                <soap:address location="mailto:quote@example1.com"/>
            </port>
            <port name="pSendResponse"
                binding="tns:ResponseSendPortBinding">
                <soap:address location="mailto:response@example2.com"/>
            </port>
            <xlang:behavior>
                <xlang:body>
                    <xlang:sequence>
                        <xlang:action
                            operation="AskLastTradePrice"
                            port="pGetRequest"
                            activation="true"/>
                        <xlang:action
                            operation="SendLastTradePrice"
                            port="pSendResponse"/>
                    </xlang:sequence>
                </xlang:body>
            </xlang:behavior>
        </service>
</definitions>
```

Code 1.4: XLANG service definition example [XLANG, 2001]

WSFL:

With IBM's WSFL [WSFL, 2001] one can model two types of (Web) Service compositions:

- Flow Model : This is a description of a business process. The model describes the usage pattern of the functionalities provided by the collection of composed Web Services. The model specifies an execution sequence.
- Global Model : This is a description of partner interactions. The model describes the interaction pattern of the collection of composed Web Services. The model does not specify an execution sequence.

WSFL allows to model recursive composition of (Web) Services. This means that once one has modeled a (Web) Service (Flow model or global model), one can use this model as a component in a new (Web) Service composition. Also WSFL has a lifecycle interface with the following operations, which makes the execution of the business process manageable:

- `spawn` : Creates a new instance of the specified flow model and starts it. The operation returns as soon as the flow model is started, passing the instance identity of the new flow instance as a result.
- `call` : Creates a new instance of the specified flow model and executes it. The operation returns as soon as the instance of the flow model has finished, passing the output of the flow instance as a result.
- `suspend` : This operation temporarily interrupts the flow instance which has the specified instance identity.
- `resume` : This operation continues the execution of the flow instance which has the specified instance identity.
- `enquire` : This operation gives back the status of the flow instance, which has the specified instance identity.
- `terminate` : Terminates the flow instance, which has the specified instance identity.

In figure 1.24 below you see an overview of the WSFL elements and where they are applied in the business process model.

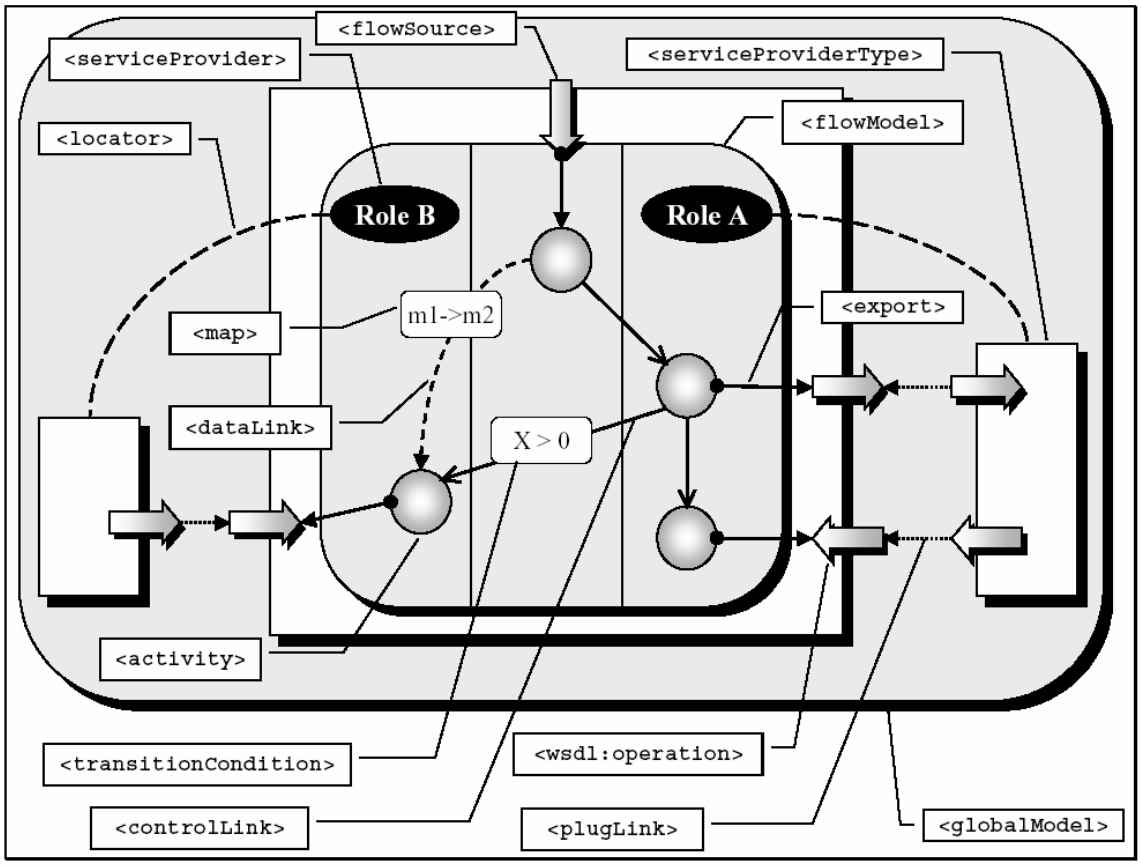


Figure 1.24: Overview of WSFL elements for a business process [WSFL, 2001].

As already stated above there are two types of (Web) Service compositions. The Flow Model is represented in WSFL by a `<flowmodel>` element. A Flow Model contains two kinds of information:

- The specification of how the composition uses the services being composed to create a flow model.
- The definition of service interface provided by the composition.

Within the Flow Model there are four parts that can be defined:

- `<serviceProvider>` : The service provider represents the endpoint that provides a set of services in the Flow Model.
- `<activity>` : The `<activity>` element specifies an activity that has to be performed in the business process.
- `<controlLink>` : A control link specifies an activity (source) and its possible successor activities (targets) in the Flow Model.
- `<dataLink>` : A data link defines the exchange of information / data between activities.

The Global Model is represented in WSFL by the `<globalmodel>` element. The Global Model defines interactions between partners in terms of client/server relationships between operations of their public interfaces. The Global Model uses a `<plugLink>` element to describe the association between “proxy” operations and the operations that are used to actually realize these proxies. A “proxy” operation is an operation that is abstractly defined for the Flow Model to be used. A plug link is used in WSFL to model the interaction between remote service providers. In this way one only needs to change the Global Model when changes occur at remote service providers that influence the provided operations.

1.2.2. BPEL4WS

As described in the introduction of this thesis BPEL4WS can be used to model formal specifications of business processes and business interaction protocols and uses the Web Services standards to enable the interaction between the modeled process and other processes / applications.

The modeling language BPEL4WS actually consists of a two components:

- BPEL4WS : An XML-based modeling language that can be used to describe the process.
- WSDL : Web Services Description Language, another XML-based language that can be used to define the types and interaction possibilities for a BPEL4WS process.

This subsection describes both languages in short. The description of WSDL contains only those elements that are of interest to BPEL4WS. In this subsection first WSDL is described and then BPEL4WS.

WSDL:

WSDL [WSDL, 2001] is an XML-based language that can be used for the definition of network services. WSDL uses the following elements for defining network services:

- Types : A container for data type definitions using some type system (such as XSD).
- Message : An abstract, typed definition of the data being communicated.
- Operation : An abstract description of an action supported by the service.
- Port Type : An abstract set of operations supported by one or more endpoints.
- Binding : A concrete protocol and data format specification for a particular port type.
- Port : A single endpoint defined as a combination of a binding and a network address.
- Service : A collection of related endpoints.

All these elements are described in detail in Section 2 of the WSDL specification, but here the first four elements are described in short because they are of interest to BPEL4WS. WSDL can be extended with new elements which definitions can be found in an associated namespace (e.g. XLANG uses this extensibility possibility of WSDL).

Types:

The default type system that is used within WSDL is XML Schema Datatypes (XSD) [XSD, 2004]. Within a WSDL file one can also define complex/structured types (other than Boolean, Integer or String, etc.), which can be used in the definition of the exchanged messages.
New type definitions in WSDL are preferably defined by XSD, but one can also use another type system. WSDL offers an element `<types>` in which zero or more type definitions can be added.
How one can define a new type with XSD is not discussed in this thesis. If one is interested in this subject, one should read the XSD specification [XSD, 2004].
Code 1.5 below shows the definition for the `<types>` element for WSDL

```
<types>
      One or more new types using a particular type system
</types>
```

Code 1.5: Definition of the <types> element for WSDL [WSDL, 2001].

Message:
In WSDL one can define zero or more abstract message definitions, which can be used in the message exchange of a process.
The definition of a message consists out of the definition of a name for the message and zero or more message parts. A message part contains a name and an element or a type, which can be defined inside the <types> element of the same WSDL file but also in another associated file containing type definitions. The value of a part of a real message must be of the same type as defined for that specific part of the related abstract message.
Code 1.6 below shows the definition for the abstract message for WSDL.

```
<message name="name">
      <part name="name" element="element"? type="type"?/>*
</message>
```

Code 1.6: Definition of the <message> element for WSDL [WSDL, 2001].

Port type and operation:
A port type in WSDL defines a set of abstract operations in which abstract messages are involved. These operations are available for a process to express its interaction pattern with another process. An operation can express one of the following interaction patterns:

1. "One-way" : The endpoint receives a message (only the <input> structure is used in the <operation> structure).
2. "Request-Response" : The endpoint receives a message, and sends a correlated message back (first the <input> structure is used, then the <output> structure).
3. "Solicit-Response" : The endpoint sends a message, and receives a correlated message back (first the <output> structure is used, then the <input> structure).
4. "Notification" : The endpoint sends a message (only the <output> structure is used in the <operation> structure).

In the interaction patterns 2 and 3 also a fault can respectively be sent or received when an error occurred in respectively the 'Request-Response' or the 'Solicit-Response' (using the <fault> structure to indicate the type of error that has occurred). In BPEL4WS only the first two interaction patterns are used.
Code 1.7 below shows the definition of a port type having two operations, the first one having a "One-way" interaction pattern and the second one having a "Request-Response" interaction pattern with zero or more faults.

```
<portType name="name">
      <operation name="name">
            <input name="name"? message="message"/>
      </operation>
      <operation name="name">
            <input name="name"? message="message"/>
            <output name="name"? message="message"/>
            <fault name="name" message="message"/>*
      </operation>
</portType>
```

Code 1.7: Definition of the <portType> element for WSDL [WSDL, 2001].

BPEL4WS:

A BPEL4WS process consists of one single BPEL4WS file. Code 1.8 below shows the default layout of a BPEL4WS file.

```
<process name="name" targetNamespace="URI"
         queryLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
         expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
         suppressJoinFailure="no" enableInstanceCompensation="no"
         abstractProcess="no"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

       <partnerLinks>?
              <!-- Note: At least one role must be specified -->
              <partnerLink name="name" partnerLinkType="partnerLinkType"
                     myRole="role"? partnerRole="role"?/>+
       </partnerLinks>

       <partners>?
              <partner name="name">+
                     <partnerLink name="partner link name"/>+
              </partner>
       </partners>

       <variables>?
              <variable name="name" messageType="message"? type="type"?
                     element="element"?/>+
       </variables>

       <correlatioSets>?
              <correlationSet name="name" properties="list of properties"/>+
       </correlationSets>

       <faultHandlers>?
              <!-- Note: There must be at least fault handler or default -->
              <catch faultName="fault"? faultVariable="fault variable"?>*
                     activity
              </catch>
              <catchAll>?
                     activity
              </catchAll>
       </faultHandlers>

       <compensationHandler>?
              activity
       </compensationHandler>

       <eventHandlers>?
              <onMessage partnerLink="partner link name" portType="portType"
                         operation="operation" variable="variable name"?>
                     <correlations>?
                            <correlation set="correlation set name"
                                   initiate="yes|no"?/>+
                     </correlations>
                     activity
              </onMessage>
              <onAlarm (for="duration-expression"|until="deadline-
                       expression")>*
                     activity
              </onAlarm>
       </eventHandlers>

       activity
</process>
```

Code 1.8: Default layout of a BPEL4WS file [BPEL4WS, 2003].

As one can see in the default layout definition of a BPEL4WS file, a BPEL4WS process consists of seven optional parts and one mandatory part:

- Partner links : Describes communication possibilities for the process.
- Partners : Describes communication restrictions for other processes with which this process communicates (not often used; therefore not used in this GA).
- Variables : Temporary storage possibilities for values that are important in the BPEL4WS process' execution.
- Correlation sets : Sets that can be used in the message correlation of the BPEL4W process.
- Fault handlers : Handlers for catching the occurrence of a fault in the BPEL4WS process' execution.
- Compensation handler : Compensation activities when the BPEL4WS process needs to be compensated.
- Event handlers : Handlers for catching events (incoming messages and / or alarms) that are related to the BPEL4WS process.
- Main activity : Describes the mandatory **activity** of the process.

An **activity** can be described by an activity element of BPEL4WS. In BPEL4WS there are two different kinds of activity elements:

- A basic activity element which describes a single activity.
  - `<receive>` : Receiving a message.
  - `<reply>` : Sending a message.
  - `<invoke>` : Sending (and receiving) a message.
  - `<assign>` : Assignment statements.
  - `<throw>` : Raise a fault within the process.
  - `<terminate>` : Abrupt termination of the process (executable process model only).
  - `<wait>` : Wait for a specified time within a process.
  - `<empty>` : Dummy activity ("Do nothing").
  - `<compensate>` : Asking for compensation of a (sub-)process.
- A structured activity element which describes an activity in which other activity elements can be defined.
  - `<sequence>` : Sequential routing of activities.
  - `<flow>` : Parallel routing of activities.
  - `<switch>` : Making a choice between activities that can be performed.
  - `<while>` : Iteration of an activity
  - `<pick>` : Receiving a message and acting upon it.
  - `<scope>` : Grouping activities (defining a sub-process).

The definition of these activity elements can be found in the BPEL4WS specification [BPEL4WS, 2003].

In BPEL4WS the following subjects are important:

- Communication.
- Data handling.
- Message correlation.

The rest of this subsection describes the subjects mentioned above.

Communication:
In BPEL4WS this is the most important subject, since BPEL4WS is mainly used to model formal specifications of business processes and business interaction protocols.

Defining communication possibilities for a BPEL4WS process involves the following steps:

1. Define the (abstract) WSDL messages that a BPEL4WS process can receive and / or send (`<message>` element of WSDL).
2. Define one or more (abstract) sets of interaction patterns using WSDL port type definitions containing (abstract) operations (`<portType>` element of WSDL).
3. Define parties (roles) which use a specific set of interaction patterns in an associated WSDL file.
4. Define the communication possibilities in the associated BPEL4WS file using partner links (`<partnerLink>` element of BPEL4WS).
5. Define for each partner link which party (role) the process represents in the communication with other processes (use of the attributes `myRole` and `partnerRole` of the `<partnerLink>` element of BPEL4WS).

Except for step 3 in the creation of communication possibilities, there are elements present in WSDL [WSDL, 2001] and BPEL4WS to define these steps. In step 3 the BPEL4WS specification [BPEL4WS, 2003] adds the following new element to WSDL: the partner link type definition.

Partner link type:
A partner link type is represented by an element `<plnk:partnerLinkType>` in the WSDL file associated with the BPEL4WS process. A partner link type represents a form of communication that the BPEL4WS process can use:

- One-party interaction : Only one role available for communication purposes.
- Two-party interaction : Two roles available (one per process) for communication purposes.

A role refers to the definition of a port type that is used for the interaction with the owner of the role. Code 1.9 below shows the definition of a partner link type for a WSDL file, where the prefix `plnk` refers to the namespace http://schemas.xmlsoap.org/ws/2003/05/partner-link.

```
<plnk:partnerLinkType name="name">
      <plnk:role name="name">
            <plnk:portType name="name of a port type"/>
      </plnk:role>
      <plnk:role name="name">?
            <plnk:portType name="name of a port type"/>
      </plnk:role>
</plnk:partnerLinkType>
```

Code 1.9: Definition of the <plnk:partnerLinkType> element for WSDL [WSDL, 2001].

The partner link type definitions can be used in the communication of a BPEL4WS process to type the communication possibilities. They also provide the roles a process can play in the interaction with other processes. The definition of a partner link in BPEL4WS is shown in code 1.10 below.

```
<partnerLink name="name" partnerLinkType="name of a partnerLinkType"
      myRole="name of a role"? partnerRole="name of a role"?/>
```

Code 1.10: Definition of the <partnerLink> element for BPEL4WS [BPEL4WS, 2003].

When an activity of a BPEL4WS process needs to communicate with another process, the activity needs to define a partner link, a port type and an operation it uses.

Oddity: Why not use the name of the role instead of the name of the port type in the specific activity that needs to communicate with another process?

In BPEL4WS the following activity elements are related to communication:

- The activity element <receive>.
- The activity element <reply>.
- The activity element <invoke>.
- The activity element <pick>.

Also the event handlers in a BPEL4WS process may use communication for the triggering of events in the process. This happens in the same way as it is done in the <pick> activity element except for the fact that a defined <pick> activity can only happen once in the execution of the BPEL4WS while the event handlers continuously listen for incoming messages to trigger events.

Data handling:
Another subject in BPEL4WS is the handling of data. BPEL4WS provides the following ways of data handling:

- The use of expressions (depends on the expressionLanguage attribute of the <process> element). Default this is XPath version 1.0 [XPath, 1999].
- The use of storage possibilities for data which has to be stored temporarily.
- Assignments to storage facilities.

The following XPath expressions can be used in BPEL4WS:

- Boolean expression : This kind of expression is indicated in a definition by "*boolean-expression*". This is an expression that results in a Boolean value.
- General expression : This kind of expression is indicated in a definition by "*general-expression*". This is an expression that results in a value that is either a number, a string or a Boolean.
- Duration expression : This kind of expression is indicated in a definition by "*duration-expression*". This is an expression that results in the value "true" when a certain time period defined by the expression has passed.
- Deadline expression : This kind of expression is indicated in a definition by "*deadline-expression*". This is an expression that results in the value "true" when a specified moment in time is reached.

BPEL4WS provides the <variable> element that can be used to define a temporary storage of data. A BPEL4WS variable must be of one of the following types:

- messageType : A WSDL [WSDL, 2001] message type.
- type : An XML Schema [XSD, 2004] simple type.
- element : An XML Schema element.

Note: An XML Schema complex type must be associated with element to be used by a BPEL4WS variable.

Finally BPEL4WS provides the <assign> activity element in which one or more <copy> sub-elements are placed which perform from-to assignments. For the exact specification of these from-to assignments one should read Chapter 9 of the BPEL4WS specification [BPEL4WS, 2003].

Message correlation:
Message correlation in a BPEL4WS process is very important when one executes the process. When one executes a BPEL4WS process, a process instance is created. To relate incoming and outgoing messages to a process instance, message correlation is needed.

Defining message correlation in BPEL4WS involves the following steps:
1. Define properties which can be used in the correlation of messages.
2. Relate each property to a part of a message that is used in the correlation of the same message.
3. Define correlation sets in BPEL4WS.
4. Define the use of correlation in the communication activities.

Only steps 3 and 4 of the above steps are already provided, that is why the BPEL4WS specification [BPEL4WS, 2003] adds the following new elements to WSDL [WSDL, 2001]: property and property alias.

Property and property alias:
A property can be used in a correlation set of a BPEL4WS process. A correlation set defines a set of properties that is used to perform message correlation in a BPEL4WS process. Message correlation is defined as relating the incoming and outgoing messages to a certain process instance during the execution of the BPEL4WS process. A property is represented by an element <bpws:property> in the WSDL file associated with the BPEL4WS process. The definition of it can be seen in code 1.11 below, where the prefix bpws refers to the namespace http://schemas.xmlsoap.org/ws/2003/03/business-process.

```
<bpws:property name="name" type="XSD (simple) type"/>
```

Code 1.11: Definition of the <bpws:property> element for WSDL [WSDL, 2001].

A property is always related to a certain part of a particular message in the message exchange of a BPEL4WS process. To provide the property element with data, an element <bpws:propertyAlias> is added to the WSDL telling which part of the message holds the value of the property. The definition of the property alias is shown below in code 1.12

```
<bpws:propertyAlias propertyName="property" messageType="message"
                    part="part of the message"
                    query="query to the part in the message"/>
```

Code 1.12: Definition of the <bpws:propertyAlias> element for WSDL [WSDL, 2001].

Once all correlation properties and property aliases have been defined for a BPEL4WS process, the correlation sets can be defined in the BPEL4WS as shown in code 1.8.

To relate a correlation set to a particular communication activity, one needs to define correlations for that activity. BPEL4WS provides the <correlations> sub-element for a communication activity to group all correlation sets that it uses. The use of a correlation set is defined as shown in code 1.13 below, where:
- The attribute initiate tells the process whether the incoming message initiates the related correlation set or not.
- The attribute pattern can only be used in an <invoke> activity element (representing a “Request-Response” interaction). The value of the attribute defines in which communication direction the related the correlation set is used for message correlation.

```
<correlation set="name of a correlation set" initiate="yes|no"?
             pattern="in|out|out-in"?/>
```

Code 1.13: Definition of the <correlation> sub-element for an activity element of BPEL4WS [BPEL4WS, 2003].

1.2.3. Process Design with BPEL4WS

In this subsection we reuse Example 1 of subsection 1.1.3. In this example there is a WS-related process which handles the “Age-verification” process for a website. In code 1.14 below the only the BPEL4WS representation of the “Age-verification” process is given.

```
<process name="AgeVerification" targetNamespace="location of this process"
         queryLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
         expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116"
         suppressJoinFailure="no" enableInstanceCompensation="no"
         abstractProcess="no"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns:def="location of the WSDL file for this process">

       <partnerLinks>
              <partnerLink name="AgeChecker"
                     partnerLinkType="def:AgeCheckingLT"
                     myRole="Checker"/>
       </partnerLinks>

       <variables>
              <variable name="AgeMsg" messageType="def:AgeRequest"/>
              <variable name="AnswerMsg" messageType="def:AccessResponse"/>
       </variables>

       <correlatioSets>
              <correlationSet name="ID" properties="def:ID"/>
       </correlationSets>

       <sequence>
              <receive name="ReceiveMessage" partnerLink="AgeChecker"
                       portType="port type belonging to the role of
                       Checker" operation="an operation belonging to the
                       port type used" variable="AgeMsg"
                       createInstance="yes">
                     <correlations>
                            <correlation set="ID" initiate="yes"/>
                     </correlations>
              </receive>

              <switch>
                     <case condition="bpws:getVariableData('AgeMsg', 'age',
                                   '/age') &lt;= 18">
                            <assign>
                                   <copy>
                                          <from variable="AgeMsg" part="ID"
                                                 query="/ID"/>
                                          <to variable="AnswerMsg" part="ID"
                                                 query="/ID"/>
                                   </copy>
                                   <copy>
                                          <from>false</from>
                                          <to variable="AnswerMsg"
                                                 part="Access"
                                                 query="/Access"/>
                                   </copy>
                            </assign>
                     </case>
```

```
                    <case condition=" bpws:getVariableData('AgeMsg', 'age',
                                '/age') > 18">
                          <assign>
                                <copy>
                                      <from variable="AgeMsg" part="ID"
                                            query="/ID"/>
                                      <to variable="AnswerMsg" part="ID"
                                            query="/ID"/>
                                </copy>
                                <copy>
                                      <from>true</from>
                                      <to variable="AnswerMsg"
                                            part="Access"
                                            query="/Access"/>
                                </copy>
                          </assign>
                    </case>
             </switch>

             <reply name="SendMessage" partnerLink="AgeChecker"
                      portType="port type belonging to the role of
                      Checker" operation="an operation belonging to the
                      port type used" variable="AnswerMsg"
                    <correlations>
                          <correlation set="ID"/>
                    </correlations>
             </reply>

       </sequence>
</process>
```

Code 1.14: BPEL4WS representation of the "Age-verification" WS-related process of Example 1.

Some final remarks for this subsection are:

Remark: For this research I have also made a definition of BPEL4WS elements and attributes in their Backus-Naur form (BNF) to understand the hierarchy inside a BPEL4WS process model. The definition of BPELWS in BNF can be found in Appendix C of this thesis.

Remark: I also made a class diagram of BPEL4WS for the same purpose as the definition of BPEL4WS in BNF. The class diagram of BPEL4WS can be found in Appendix D of this thesis.

Chapter 2: A GML for generating BPEL4WS models

2.1. Introduction

A Graphical Modeling Language (GML) is a modeling language that has a graphical notation. With help of this graphical notation it is easier for a developer to model a (business) process in a modeling language. Many (business) process modeling tools use the graphical notation of a modeling language to speed up modeling.
The language High-level Petri Nets (HLPN) [HLPN, 2000] also has a graphical notation (described in Chapter 1 of this thesis), which can be used by a developer who has knowledge of HLPN to model a process in this language.
The language BPEL4WS [BPEL4WS, 2003] on the other hand does not have a graphical notation. Several approaches have been made to create a uniform graphical notation for the language BPEL4WS[13]. In this research the language BPEL4WS had to be expressed using the semantics of the GML of HLPN.

The following approach has been taken to develop a GML that is based upon HLPN for generating BPEL4WS models:

- Translate all the BPEL4WS elements (with all attributes and all sub-elements) to their equivalents in the GML of HLPN.
- Describe these translations in EPNML [EPNML, 2004], which is an XML-based language that can be used to describe the graphical representation of a HLPN model.
- Create a translation model that can be used to translate a BPEL4WS process to a HLPN (business) process in EPNML.
- Create a mapping model based upon the translation model that offers handles (a set of HLPN elements that covers a certain activity in the process) to the developer of a WS-related process model.

The first three topics above involve the creation of the Translation Model, which is described next. The Mapping Model is described later in this chapter.

2.2. Translation Model

The Translation Model (hereafter abbreviated as *TM*) is a model that has been created to translate the contents of a BPEL4WS process to its equivalent in the GML of HLPN and EPNML. The TM translates each BPEL4WS element and WSDL element of a BPEL4WS process to a standard pattern in the GML of HLPN and EPNML. The TM also describes the coupling between these standard patterns. The result of a translation using the TM is an EPNML file in which some small extensions have been made to the EPNML standard (described later in this subsection). An EPNML file can be opened in the tool that supports the EPNML standard [EPNML, 2004]. YASPER[14] is such a tool. In YASPER one does not see all the information that is present in the resulting EPNML file of the translation, because YASPER currently does not support the extensions made to the EPNML standard in this GA. However the GA's PN-BPEL4WS Tool (described in Chapter 3 of this thesis) does show this information. The TM can be found in Appendix E of this thesis.
In the rest of this subsection the following topics concerning the TM are discussed:

- Creating the TM.
- An example using the TM.

[13] The tool Oracle BPEL Designer (http://otn.oracle.com/bpel) offers graphical representations of BPEL4WS elements to speed up the modeling in this language.
A study by Daniela Weinberg (http://www.informatik.hu-berlin.de/top/download/publications/weinberg03.pdf) also offers graphical representations of BPEL4WS elements that can be used to model a process (as far as I know there is no tool that supports this GML by Weinberg).
[14] http://www.yasper.org

2.2.1. Creating the Translation Model

As described in the introduction of this chapter, the first step to the creation of the TM was the translation of all BPEL4WS elements with all possible attributes and all possible sub-elements. To start with this translation, a point-of-view had to be taken considering the modeling of a business process. In my opinion there are two kinds of modeling:

- Looking at all communicating processes together and considering them as one process that has to be translated.
- Looking at each communicating process in the business process separately, which (when all translated) must be coupled (= compositional).

I use the second kind of modeling, because BPEL4WS also uses this kind of modeling where the (actual) coupling between the separate BPEL4WS processes is defined in the associated WSDL file(s). When a process in the business process model changes, one only has to change that process model and not the whole business process as would be the case in the first kind of modeling provided that the communication stays the same between the changed process and the other processes in the business process model.

The first approach to translate the BPEL4WS elements to their graphical HLPN form was a rough translation of the initial example of the BPEL4WS specification ([BPEL4WS, 2003], subsection 6.1). The source of this example can be found in Appendix G of this thesis. The result of this rough translation is shown in figure 2.1 below.

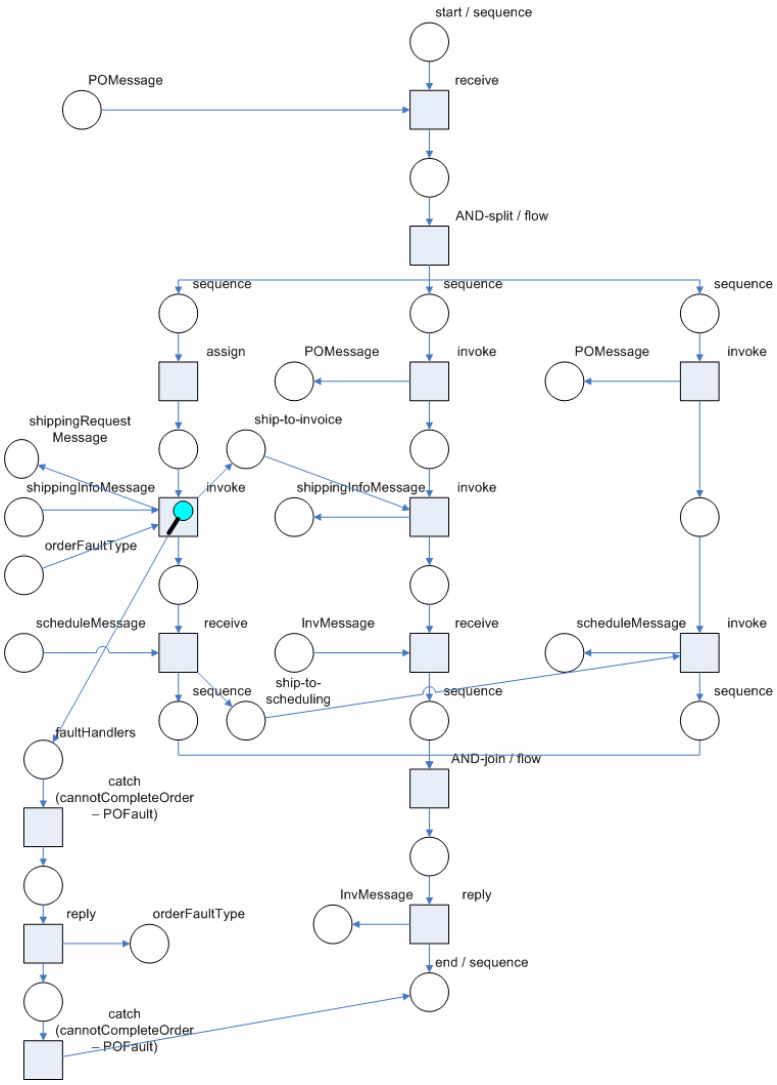


Figure 2.1: Initial example on page 16 of the BPEL4WS specification in its rough graphical HLPN form.

One might notice in figure 2.1 that the variables and partner links of the initial example of the BPEL4WS specification have not been modeled. A variable could have been modeled by a single place in which a token representing the variable resides. All activities that use this variable then should be coupled to this place. I left it out of the figure above to improve the readability of the model. The partner links proved to be a difficult item to model in this example. In this case I only provided the kind of message that was received / sent. Finally I managed to come up with a translation of a partner link (see subsection 2.2.1 ”Communication”).
In figure 2.1 one can already see some special places, namely:

- Message places (e.g. “POMessage” and “InvMessage”).
- Place for handling the occurrence of a fault in the process (e.g. “faultHandlers”).
- Places that represent a link (e.g. “ship-to-invoice”).
- Places and transitions that belong to the main process.

These special places play an important role in the readability and coupling of the models.

In the translation of elements from one language to another language one always has to look carefully at the semantics of each language that is involved in the translation. An example in our case is that when one tries to translate a source file of a BPEL4WS process one-to-one to its graphical representation in the GML of HLPN without looking carefully at the semantics of BPEL4WS, one might overlook the situation in which a fault handler is omitted in a particular scope. The semantics of BPEL4WS describe that there actually is a “stealth” fault handler present that catches the fault and terminates the process abnormally.

In this subsection the following topics concerning the translation of the BPEL4WS elements are described:

- General design decisions : Decisions that were made to be able to translate BPEL4WS elements to their HLPN equivalent in EPNML.
- Graphical notation : Description of the graphical notation used in the TM.
- Approach for the translation : Description of the approach that has been taken when translating a BPEL4WS process model.
- Element translation : Description of the translation of BPEL4WS elements.
- Process translation : Description of the translation of a process.
- Communication : Description how the communication is handled in the translation of a BPEL4WS process.
- Process instance translation : Description of the translation of process instances.
- Start activities : Description of the translation of the “start activities” in a BPEL4WS process.
- Routing : Description how the routing of activities can be translated.
- Event handling : Description how the event handlers are translated.
- Fault handling : Description how the fault handlers are translated.
- Compensation handling : Description how the compensation handler is translated.
- Termination : How process termination works in the translation of the process.

General design decisions:

When I looked at each element of the BPEL4WS language separately, I made some design decisions that helped me to translate each element. The design decisions are displayed in table 2.1 below.

Design Decision	Comment
Creation of token types	In HLPN [HLPN, 2000] this feature already exists, but in EPNML it is not possible to relate a token to a type. This is an extension to the EPNML standard where a token type definition represented by the XML tag <token> can be added to the net's definition in the EPNML file. A token type has a certain type, which in this GA can be: • ***process*:** This kind of token type contains information about the process instance. Only one can be defined per <net>. • ***fault*:** This kind of token type contains information about a fault that can occur in the process. Zero or more can be defined per <net>. • ***message*:** This kind of token type contains a representation of a message that can be received / sent by the process. Zero or more can be defined per <net>.
Add token type restrictions to places	In HLPN [HLPN, 2000] this feature already exists, but in EPNML it is not possible to add this kind of restriction to a <place> element. That is why an extension to the EPNML standard has been made to include this feature. It is represented by the XML tag <tokenType> as a sub-element of the <place> element. Adding a <tokenType>-tag to a place restricts the place from receiving any other token than that of the token type defined by the <tokenType>-tag for the place.
All variables are added to the token type definition of the process	This to improve the readability of the process model. Also this provides a simpler way to update the value of a variable in an activity (transition).
All correlation sets are added to the token type definition of the process	This to improve the readability of the process model. Also this provides a simpler way to update the value of a correlation set in an activity (transition), and can be used in the message correlation.
All links are added to the token type definition of the process	This to improve the readability of the process model. Activities that depend on the status of a link now are able to read it from the process instance. Activities also can alter the status of a link.
Goal of reset-arcs in the model	In a model of this GA a reset-arc between a transition and a place has the following meaning: The transition should only consume from that place the tokens that represent the same process instance of the token that is used in the firing of the transition.
Use of YASPER's collector and emitor for the representation of sources and sinks in the model.	YASPER provides a transition that can serve as a source for tokens, the emitor, and a transition that only eats tokens and does not produce any, the collector (a sink). In our model we use the emitor to generate a message token for a communication port that is not connected to another process, and the collector to collect "left behind" tokens in the process (e.g. when an alarm has been handled).
Creation of a store to maintain all process instances	In parallel activities it proved to be difficult to model a process instance with the most recent process instance values. By modeling a store that contains all process instances of a certain type this problem is solved (store must use the <tokenType>-tag containing the token type of the process). Activities now can use this store to create, read and update process instances.

Table 2.1: Design decisions when translating BPEL4WS elements to their HLPN equivalent.

Design Decision	Comment
Adding color to places to specify special places	In the HLPN translations of BPEL4WS elements there is a need for defining special places (e.g. for fault handlers of a process). Each special place is indicated by a certain color that is in conformance with CSS2 [CSS2, 1998] as is defined in the PNML definition [PNML, 2003]. A place can get a color via the `<graphics>` sub-element. To this element the `<fill>` element is added of the PNML definition, thus is an extension to EPNML. The following colors are defined for my translation: • ***yellow*****:** The place belongs to the main activity of the process. • ***red*****:** The place belongs to a fault handler. • ***green*****:** The place belongs to an event handler. • ***orange*****:** The place belongs to a compensation handler. • ***purple*****:** The place is a communication port (hereafter called *COM-port*). A COM-port in this model is a communicating connection to another (participating) process in the business process model. • ***black*****:** The place belongs to the termination of the process.
Urgent transitions	Urgent transitions in the TM are transitions that have either a **black** or **red** input place. These transitions must fire immediately when they are enabled. The urgent transitions are used in the **termination** (**black** input place) / **halting** (**red** input place) of a process. Also there is a priority between the firing of these urgent transitions, namely: urgent transitions that have a **black** input place fire before urgent transitions that have a **red** input place.

Table 2.1 (continued): Design decisions when translating BPEL4WS elements to their HLPN equivalent.

Having made these decisions I was able to translate the BPEL4WS elements to their HLPN equivalent in EPNML.
But before I start to describe the rest, I need to discuss these (new) extensions (see [YASPER, 2005] for the definitions of the emitor and collector).

The token type definition (`<token>`):
This new XML tag can be used within a `<net>` element of the EPNML standard. It specifies a token type definition. A token type definition exists of two main items: its type and its contents. The token type definition is defined as described below.

```
<toolspecific tool="PN-BPEL4WS" version="1.0">
      <token type="process|fault|message" id="uniqueID">
            Contents of the token
      </token>
</toolspecific>
```

Code 2.1: Definition of a token type.

As we have seen in table 2.1 there are three types available: *process*, *fault* and *message*. The first of these types, *process*, describes a process instance of the process model. The process instance can contain any data that is relevant to the process. In our case we add the following information to our process instance to be able to maintain the information of a BPEL4WS process:

- Settings : The settings in the `<process>` element of BPEL4WS that need to be preserved (e.g. references to external definitions).
- Variables : A variable contains a value that is needed by an activity in the process. The value of this variable may alter during the run-time / execution of the process.

- Correlation sets : A correlation set contains a list of values (one or more) that is needed to distinguish the process instances amongst each other. A correlation set is initialized once and cannot be reset / altered elsewhere in the process. Correlation sets are also used during run-time / execution to check if the incoming message belongs to the process instance.
- Links : A link is in fact a sort of control (to be used in a concurrent / parallel execution of activities). Depending on the status of a link, a certain activity that uses this link may be executed / skipped or may cause a fault to occur. A link must be initialized before it can be used by an activity. Activities may initialize and alter the status of a link, but once it has the status "true", it will not change anymore.

The definition of a token type of this kind is shown below.

```
<token type="process" id="process_name of the process">
      <name>name of the process</name>
      <location>location of the process</location>
      <queryLanguage>?
            location of the query language's definitions
      </queryLanguage>
      <expressionLanguage>?
            location of the expression language's definitions
      </expressionLanguage>
      <suppressJoinFailure>
            yes|no
      </suppressJoinFailure>?
      <enableInstanceCompensation>?
            yes|no
      </enableInstanceCompensation>
      <abstractProcess>
            yes|no
      </abstractProcess>?
      <namespaces>?
            <namespace id="name">"location"</namespace>+
      </namespaces>
      <variables>?
            <variable name="name" type="type" kind="kind"/>+
      </variables>
      <correlationSets>?
            <correlationSet name="name">+
                  <list>
                        <item name="name" type="type"
                              message="tokenName" part="part"
                              query="query"/>+
                  </list>
            </correlationSet>
      </correlationSets>
      <links>?
            <link name="name">+
                  <initialized>true|false</initialized>
                  <status>true|false</status>
            </link>
      </links>
</token>
```

Code 2.2: Definition of a token type of the kind 'process'.

The second type, *fault*, describes a fault that can occur during the execution of the process. In BPEL4WS a fault can consist of two parts: a name of the fault and a variable to be associated with the fault that occurred. This variable must be present in the process instance. The definition of a token type of this kind is shown below.

```
<token type="fault" id="fault_name and / or variable">
      <name>?
            name of the fault
      </name>
      <variable>?
            name of the variable
      </variable>
</token>
```

Code 2.3: Definition of a token type of the kind 'fault'.

The third type, message, describes a message that can be received / sent by an activity of the process. This token type is used in the communication of the process with other (participating) processes in the business process model. A message has a certain name and can contain data described by the <part>s. The definition of a token type of this kind is shown below.

```
<token type="message" id="message_name of the message">
      <name> name of the message </name>
      <parts>?
            <part>+
                  <name>name</name>
                  <kind>kind</kind>
                  <type>type</type>
            </part>
      </parts>
</token>
```

Code 2.4: Definition of a token type of the kind 'message'.

Token type restrictions for places (<tokenType>):
A place may contain a token type restriction. This has the following consequence for a place in the model: it can only receive tokens of the kind defined by the <tokenType> element. To add a token type restriction to a place one simply adds the following piece of code between the <place>-tags.

```
<toolspecific tool="PN-BPEL4WS" version="1.0">
      <tokenType>Name of a token type</tokenType>
</toolspecific>
```

Code 2.5: Definition of a token type restriction inside a place.

Coloring of places (<fill>):
As described in table 2.1, I make a distinction between six different kinds of places. To show this distinction graphically I added a color to a place. The PNML definition provides a <fill> element that can be used inside a <graphics> element to describe the internal color of a place, transition or arc. I use this element for the definition of special places in my model. The following piece of code has to be inserted between the <graphics>-tags of a place.

```
<toolspecific tool="PN-BPEL4WS" version="1.0">
      <fill color="CSS2-color">
</toolspecific>
```

Code 2.6: Definition of the <fill> element for a place.

Graphical notation:

Table 2.2 shows the graphical notation that is used in my translation of BPEL4WS elements to their equivalent in HLPN.

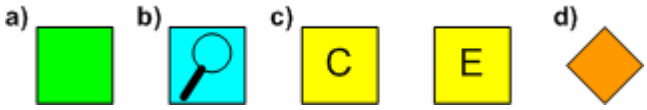
Element	Graphical notation
Places	 Place –types (type *channel* in EPNML): *a)* ***yellow*****:** The place belongs to the main activity of the process. *b)* ***red*****:** The place belongs to a fault handler. *c)* ***green*****:** The place belongs to an event handler. *e)* ***orange*****:** The place belongs to a compensation handler. *f)* ***purple*****:** The place is a communication port (hereafter called *COM-port*). A COM-port in this model is a communicating connection to another (participating) process in the business process model. *g)* ***black*****:** The place belongs to the termination of the process. A transition that has an input place of this type is an urgent transition.
Transitions	 Transition-types: *a)* ***Normal*****:** The transition has type AND. *b)* ***Sub-process*****:** The transition in fact is a sub-process. One can zoom in on this sub-process to view its contents. *c)* ***Collector & Emitor:*** The transition is of the kind collector (C) or emitor (E) as described in [YASPER, 2005]. *d)* ***XOR*****:** The transition has type XOR.
Tokens	A token is represented by a dot inside a place. Depending on the type of place it belongs to it also has a special meaning: • ***yellow*** **(Process) and** ***black*** **(Termination):** token contains process data. • ***purple*** **(COM-port):** token represents a message. • ***red*** **(Fault):** token contains fault data. • ***green*** **(Event):** token contains process data. • ***orange*** **(Compensation):** token contains a snapshot of the process instance when the scope of the compensation handler completed. **Note:** Each process instance has its own token-color.
Stores	 This represents a place (type *store* in EPNML) that can be one of the following types: • Name is “PI”: The process instances of a process. • Name is “Calendar”: The current time in the process.

Table 2.2: Graphical notation that is used in the translation.

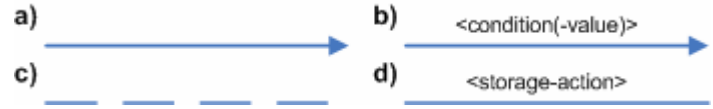
Element	Graphical notation
Arcs	 Arc-types: *a)* ***Normal*:** The arc is a normal connection between a place and a transition or between a transition and a place. *b)* ***With condition*:** This kind of arc can only be used as an output arc of a transition with type *XOR*. This type of arc displays a possible outcome of the condition of the *XOR*-transition. If the outcome of the condition is equal to that of the arc, the flow continues in the direction of that arc. *c)* ***Reset-arc*:** This type of arc is used to indicate the reset-arc as described earlier. *d)* ***Storage*-action:** This kind of arc represents a storage action as described in the EPNML standard.

Table 2.2 (continued): Graphical notation that is used in the translation.

The graphical representations of the BPEL4WS elements in the rest of this subsection have been made with YASPER and have been colored afterwards to display the extensions. This is also the reason why there are multiple reference places pointing to the same place in a parent scope; actually only one reference place should be there, but YASPER cannot handle this kind of construction. The figures give an indication how the graphical representation of a BPEL4WS element should look.

Approach for the translation:

The first step in the creation of the TM is to have a clear approach for the translation of a BPEL4WS process. My approach can be described by the following steps:

1. Take the BPEL4WS file of the process one wants to translate.
2. Retrieve the associated WSDL files that are referred to in the BPEL4WS file.
3. Check if the elements are in conformance with the BPEL4WS specification.
4. Start the translation by creating a new EPNML file.
5. Translate the <process> element.
6. Translate all communication ports.
7. Create the process-token type, message-token types and fault-token types.
8. Translate the fault handlers
9. Translate the event handlers
10. Translate the compensation handler
11. Translate the main activity
12. Check the coupling of all activities concerning the termination / halting of the process.

This approach shows a hierarchical approach of the translation, which starts at the start of the process (<process> element) and works its way down to the atomic activities in the BPEL4WS process.
The resulting TM can be found in Appendix E of this thesis. The TM uses the EPNML code of the translations to create an HLPN process model from a BPEL4WS file with its associated WSDL file(s).

In the rest of the subsection 2.2.1 the steps 5 to 12 of the approach are described in more detail. Before I start with this description I need to describe how an element is translated in my translation.

Element translation:

When creating the TM I realized that I needed a uniform way to translate the BPEL4WS (activity) elements. This uniform way of translation has been achieved through the use of hierarchical transitions and *compositional modeling*.

Each BPEL4WS (activity) element has its own characteristics as can be seen in the TM, but they all are translated using the same basis. The basis of the translation of a BPEL4WS (activity) element consists of the following construct as displayed in figure 2.2 below:

- A hierarchical transition : Inside a hierarchical transition the working of the BPEL4WS (activity) element is defined (this may include translations of other BPEL4WS (activity) elements).
- A "previous" place : A "previous" place is a place that enables the hierarchical transition's firing.
- A "next" place : A "next" place is a place that serves as an output place for the hierarchical transition.

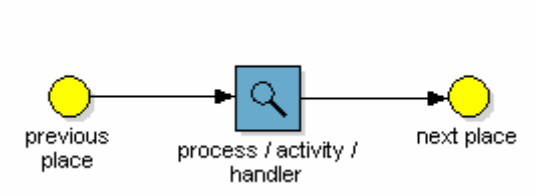


Figure 2.2: Main construct for a BPEL4WS element in the TM.

In a BPEL4WS file (see subsection 1.2.2) a BPEL4WS (activity) element is either placed after the definition of another BPEL4WS (activity) element or is placed inside the definition of another BPEL4WS (activity) element. This is why I use compositional modeling and hierarchical transitions.

Except for the basic BPEL4WS activity element's translations one needs to zoom in on the hierarchical transition of a BPEL4WS (activity) element's translation to define another BPEL4WS (activity) element's translation inside it.

Depending on the semantics of a BPEL4WS (activity) element and the position in a BPEL4WS file, the translation of that BPEL4WS (activity) element may be coupled to the translation of another BPEL4WS (activity) element. The general coupling of two BPEL4WS (activity) element's translations that are defined after each other is shown in figure 2.3 below and the general coupling of the translation of a BPEL4WS (activity) element that is defined inside another translation of a BPEL4WS (activity) element is shown in figure 2.4 below.

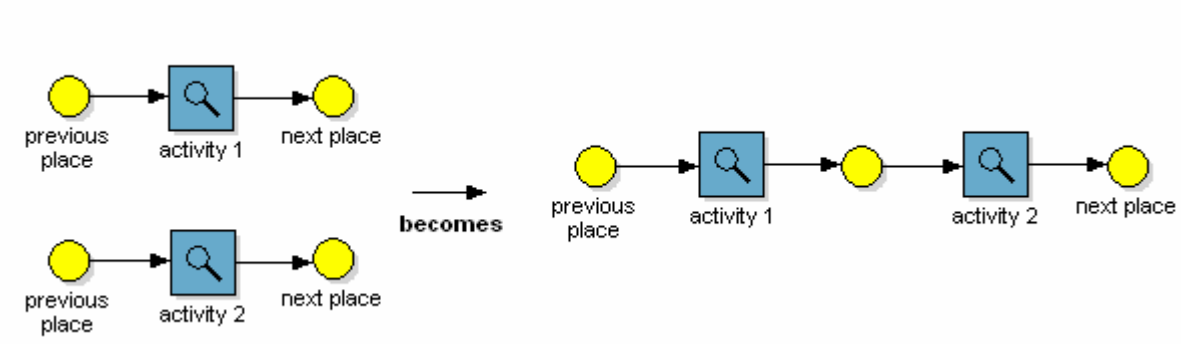


Figure 2.3: Coupling between two translated activities that are defined after each other.

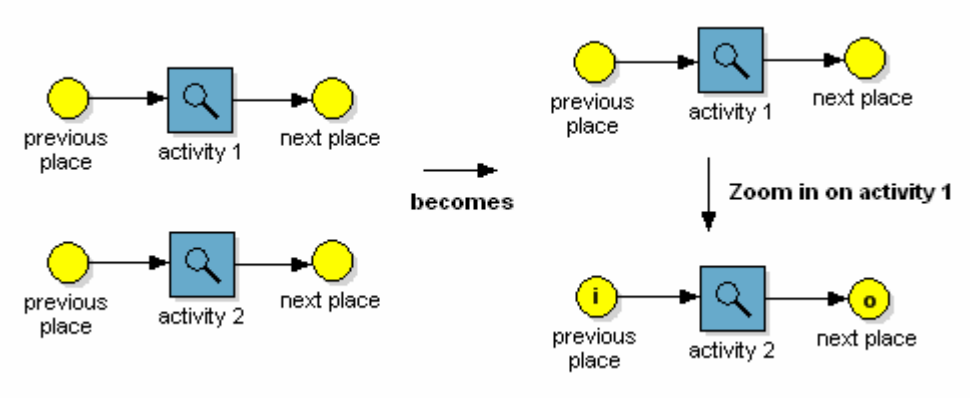


Figure 2.4: Coupling between two translated activities of which the second one is defined inside the first one.

Depending on the BPEL4WS (activity) element that is translated, one has to add places concerning communication, start of event handling, fault handling, compensation handling, creation / reading / updating of process instances, reading the calendar and / or process termination.

Process translation:

The first BPEL4WS element that one encounters in a BPEL4WS process is the `<process>` element. This element indicates the start of a new BPEL4WS process. This is also the start of my translation of a BPEL4WS process in the TM.
A BPEL4WS process is initiated when it receives a message in one of its *start activities* (defined later in this subsection) and is terminated when one of the following occurs [BPEL4WS, 2003]:

- The main activity completes in a normal way. In this case *termination* is normal.
- When a fault reaches the process scope. In this case termination is abnormal.
- When a process instance is explicitly terminated by a `<terminate>` activity element. In this case termination is abnormal.
- If a compensation handler is specified for the business process as a whole, a business process can be compensated after normal completion by platform-specific means (enabled by setting the `enableInstanceCompensation` attribute of the `<process>` element to the value “yes”).

In my translation of a process I have added the following extra features to support the business process manager:

- Activation of a process as a whole (“Activate” transition in figure 2.5).
- Termination of the process as a whole provided that the running processes can complete normally (“Stop” transition in figure 2.5).

A new process is represented by a new `<net>` in my translation. A new `<net>` called “A” without platform-specific compensation and any communication possibilities in my translation is represented by the following graphical notation (see figure 2.5).

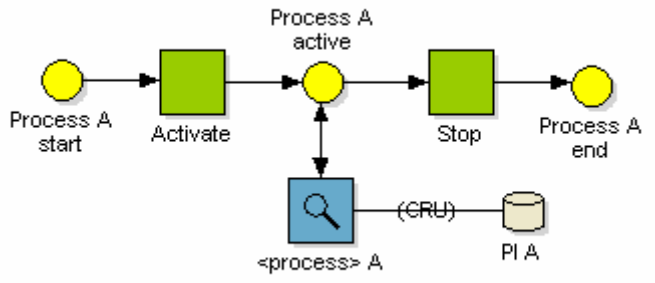


Figure 2.5: Process “A”, starting the translation.

In a BPEL4WS process one can distinguish four processes, of which initially one is active and the three others are dormant:

- **Main activity** : This is the activity that has to be performed in the process. Once a new process instance has been created, also the event handlers of the process' scope are actived for that process instance.
- **Fault Handler(s)** : This sub-process is dormant and can only become active when a fault occurs in the main activity. It stops the execution of the main activity and continues its fault handling capabilities / activities.
- **Event Handler(s)** : This sub-process is dormant and becomes active for a process instance as soon as the process instance is created. It handles events (e.g. alarms).
- **Compensation Handler**: This sub-process is dormant and becomes active when the process asks for a compensation of the scope the compensation handler belongs to. This can only happen in a compensation handler of a higher scope or in a fault handler of the same or higher scope. It then performs compensation activities (e.g. 'undo'-actions). All activities of the scope of the compensation handler must have completed normally.

Figure 2.6 zooms in on the hierarchical transition of a process like that of figure 2.5 for which event handling (EH A), fault handling (FH A) and platform-specific compensation handling (CH A) has been defined. In this figure one can clearly see the four sub-processes described above. Also in this figure the communication ports for the input and output of the Web Services related process are shown; yet the calendar and the use of process instances in the handlers are not shown, but can be present.

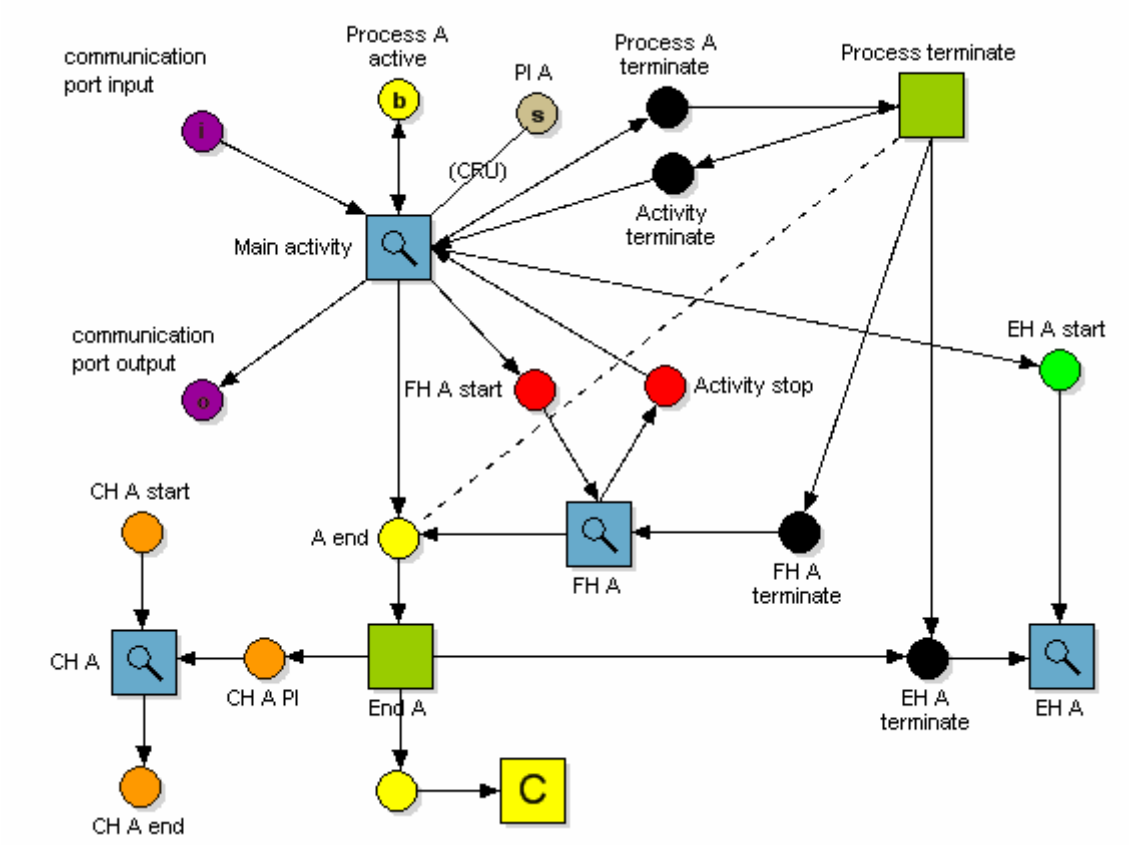


Figure 2.6: The four sub-processes of a BPEL4WS process and their relations.

When the process given in figure 2.6 is simulated, the process starts with two tokens:

- A *process* token in the place “Process A active”, which represents a process control for the business process manager.
- A *message* token in the place “communication port input”, which represents the place of incoming message that initiates a new instance of the process.

The next step in the simulation is to look in the hierarchical transition of the Main activity and follow the path from there on. There are three possible endings for this path:

- If the Main activity completes normally, then a message is sent back to the desired process(es), the compensation handler of the process gets a snapshot of the process instance upon completion of the process (CH A PI) and the event handler for the process instance is terminated.
- If inside the Main activity a fault has occurred then the fault handler (FH A) for the process is invoked, which first stops all activities inside Main activity and after that handles the fault and ends the process instance.
- If inside the Main activity an activity calls for the termination of the process, then the urgent transition “Process terminate” fires and the process instance ends abnormally.

Communication:

The next step is to define the communication for the process using COM-ports, which are special places for the message exchange between the processes of a business process. The definition of the COM-ports in BPEL4WS starts with the `<partnerLink>` elements. In my translation if a process needs to communicate with other processes, it uses these COM-ports to receive or send messages. The translation of this element also needs the WSDL file in which the partner link type of the partner link is defined. The partner link type defines one or two port types (depending on the number of roles defined), which are used to exchange messages between the process and other processes. This is where the translation for a COM-port starts. One can have the following two kinds of communication patterns in a BPEL4WS process:

- **“One-way”** : An activity in a process only sends a message to another process in the business process.
- **“Request-Response”** : An activity in a process sends a message (request) to another process and the other process eventually sends a message (response) back.

In figure 2.7 below the COM-ports are described from two angles:

- **My role (***fig 2.7 a)***)** : This angle describes that the COM-ports belong to my process. In case the interaction pattern is a “One-way” interaction pattern, then only the COM-port describing the “input” is used. In the other case also the “output” and “fault” COM-ports are used.
- **Partner role (***fig 2.7 b)***)**: This angle describes the opposite situation. In this situation the COM-ports belong to a process with which my process communicates.

Note: The incoming messages have to be copied for the main process and each event handler that is in use by the process. The emitor and collector are there to respectively generate and eat tokens when a process is not coupled to another process in the same business process. Also note the arc with the black dot at the end; this is called an *inhibitor arc*, which has the following meaning: the transition can only fire when the place connected to it has zero tokens residing in it.

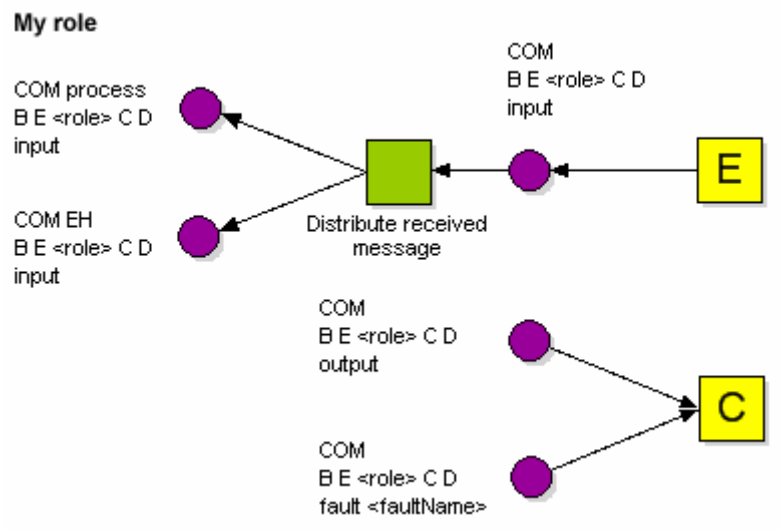


Figure 2.7: a) COM-ports belonging to my process.

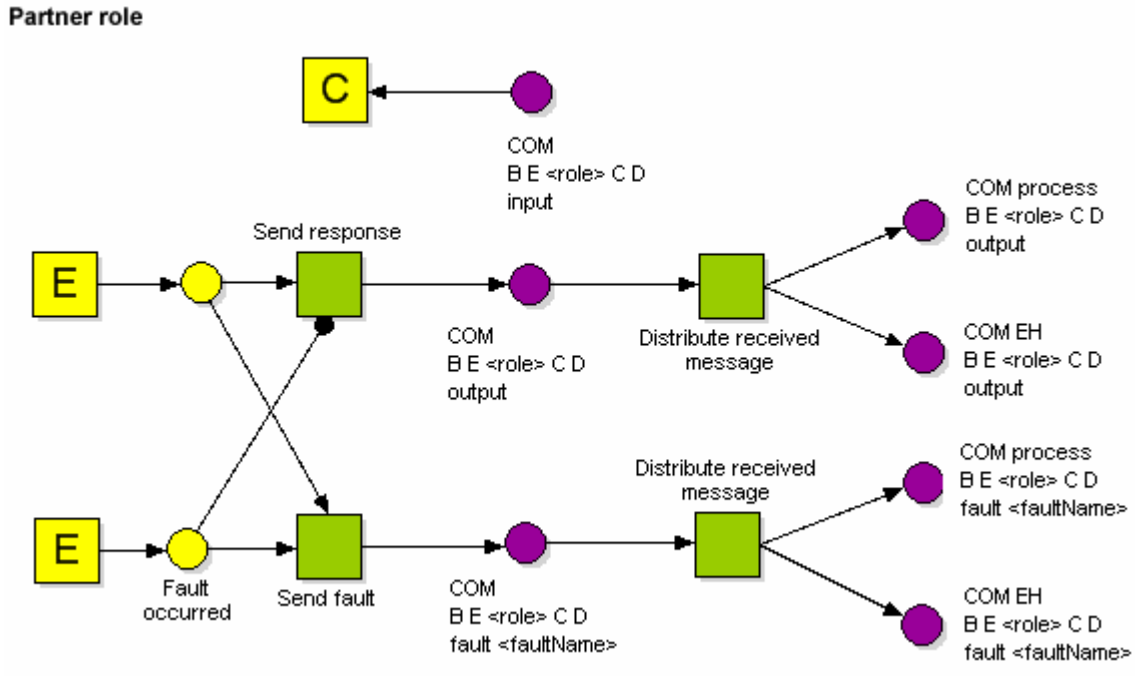


Figure 2.7: b) COM-ports belonging to a process with which my process communicates.

In figure 2.7 a) and in figure 2.7 b) some parameters are used in the definition of the COM-ports, these are:

- **B** : The name of the partner link.
- **E** : The name of the partner link type.
- **<role>** : The role the belonging to the process (`myRole`) or belonging to the other process (`partnerRole`).
- **C** : The name of the port type that is associated with the role.
- **D** : An operation of the port type.
- **<faultName>** : The name of the fault that is sent back.

Process instance translation:

The third step is to define the process instance (see 2.2.1 “The token type definition” for the contents of a process instance) and a “calendar” for the process. Once all information has been collected from the BPEL4WS process, a token of the type *process* can be created containing this information. Then a place the type *store* has to be created of (as the place “PI A” of figure 2.5), which has a `<tokenType>` sub-element that refers to the token type just defined. This kind of place, a database for process instances, can be used by activities (transitions) in the process to perform the following actions:

- **C** : Create a new process instance in the database with default data.
- **R** : Read the contents of a certain active process instance present in the database.
- **U** : Update data values of a certain active process instance present in the database

These actions can be combined, e.g. ‘CRU’ would have the following meaning: An activity creates a process instance, reads and updates data from one or more process instances.
The place “calendar” is also a place of the type *store*, where only the “read” actions can be used for retrieving the current time in the process.
Figure 2.8 below shows the above described actions on the storage for process instances and a calendar.

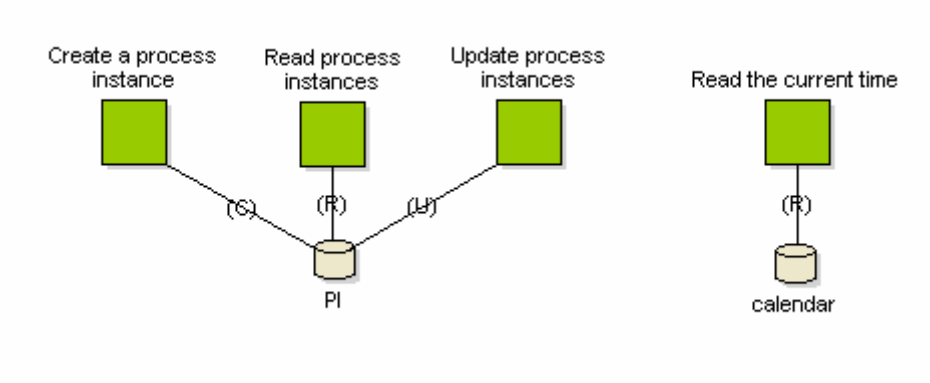


Figure 2.8: Available actions for the places “PI” and “calendar” in a process.

Fault handling:

The next step in the translation of a BPEL4WS process in the TM is the translation of the fault handlers of the process’ scope.
Fault handlers in a BPEL4WS process are grouped by the `<faultHandlers>` element. This element contains the possibility to catch a fault that has occurred during the execution of the main process. As soon as a fault occurs the fault handler of that scope stops all activities of that scope (place “End scope E” in the translation).

A `<faultHandlers>` element can have a fault handler defined, `<catch>`, which takes care of a particular fault. The handling of the fault may include compensation and / or re-throwing the fault to the fault handlers of the parent scope. One can also define a fault handler catching all faults that occur in the scope, the `<catchAll>` sub-element. If there is no `<catchAll>` sub-element used then BPEL4WS introduces a “stealth” default fault handler for that scope, which re-throws the fault to the parent scope unless the scope already is the top scope of the process.

In figure 2.9 below a translation is given which represents the situation when the definition of a <faultHandlers> element is omitted for a particular scope. As defined in the BPEL4WS semantics for the fault handling in a scope ([BPEL4WS, 2003], subsection 13.4) a "stealth" fault handler is added to the scope to handle the faults that occur inside the particular scope. This "stealth" fault handler also needs to stop all the activities inside the scope and after that check whether the scope it belongs to is the scope of the process (<process> element) or of a sub-process (<scope> element). In the first case the process ends in the latter case the fault is thrown to the parent scope.

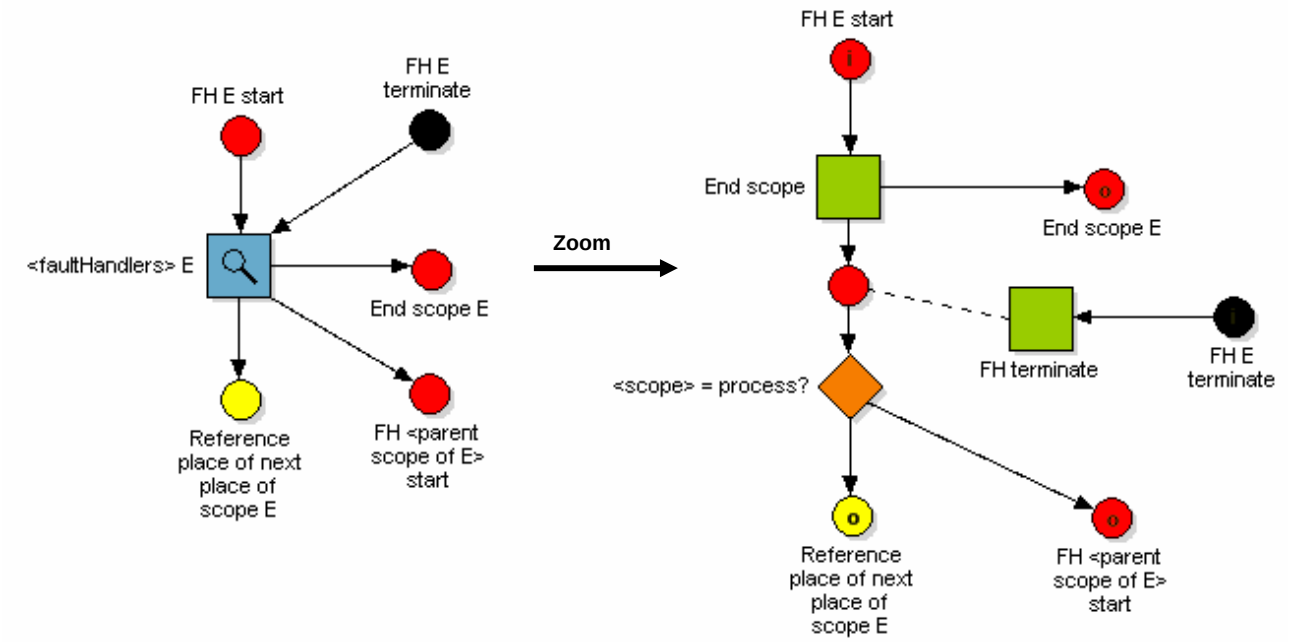


Figure 2.9: The translation when the <faultHandlers> element is omitted in the definition of a scope E.

In figure 2.10 below shows the translation of a <faultHandlers> element where one <catch> sub-element has been defined to catch a fault with name **B** and which has an associated variable **C**. This means that the definition of a <catchAll> sub-element is omitted. As defined in the BPEL4WS semantics for the fault handling in a scope ([BPEL4WS, 2003], subsection 13.4) a "stealth" fault handler is added to the scope to handle the faults that cannot be caught by the defined fault handlers for the particular scope.

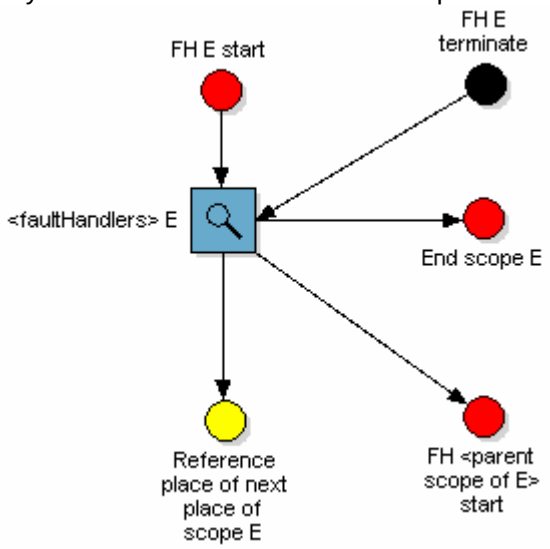


Figure 2.10: The translation of a <faultHandlers> element which contains a <catch> sub-elements and where the <catchAll> sub-element is omitted.

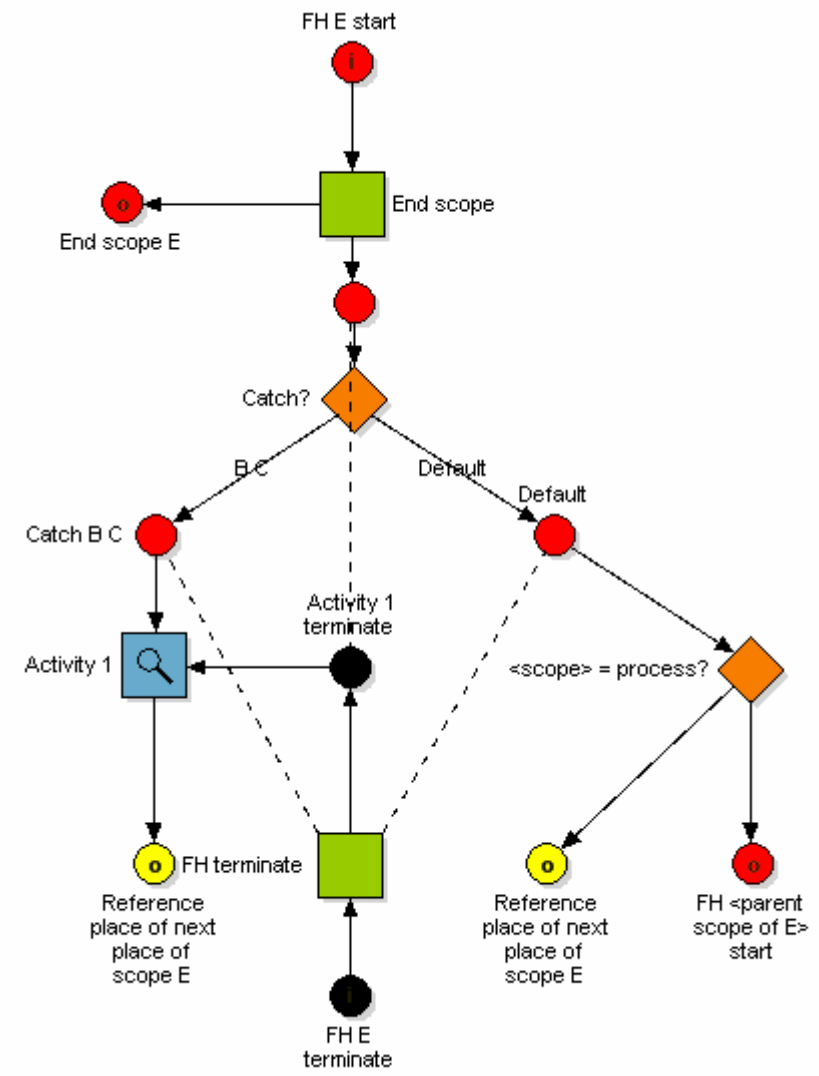


Figure 2.10 (continued; zooming in on the hierarchical transition): The translation of a <faultHandlers> element which contains a <catch> sub-elements and where the <catchAll> sub-element is omitted.

In figure 2.11 below the translation is shown of the same situation as in figure 2.10 except for the fact that this time a <catchAll> sub-element is present in the definition of the fault handlers. This also means that the fault cannot be thrown to a parent scope unless a <throw> activity element is used inside a fault handler's definition.

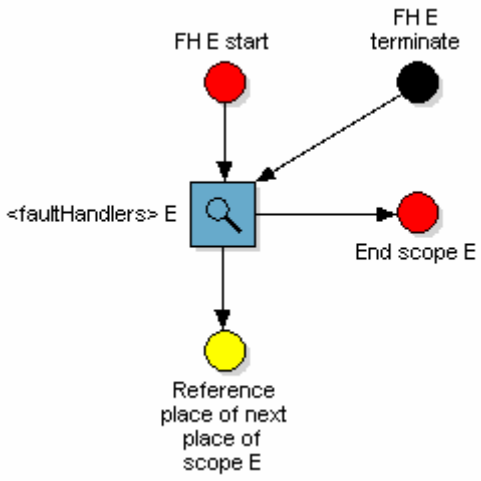


Figure 2.11: The translation of a <faultHandlers> element which contains a <catch> sub-elements and where the <catchAll> sub-element is used.

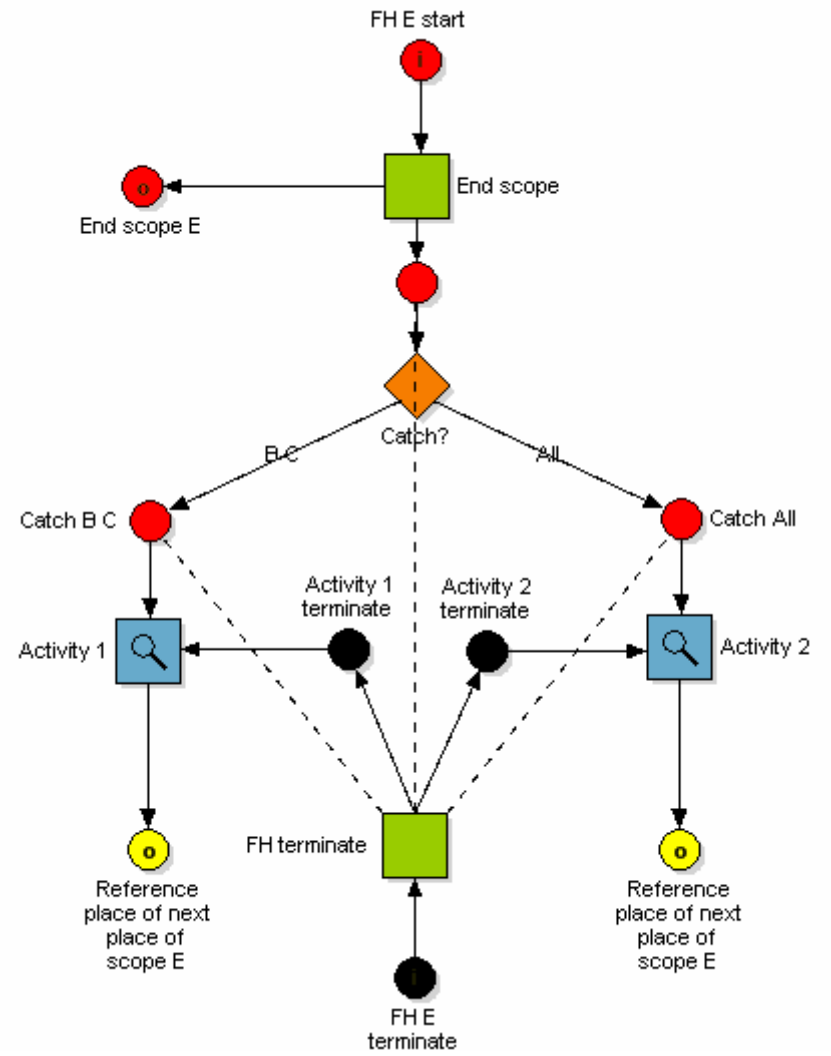


Figure 2.11 (continued; zooming in on the hierarchical transition): The translation of a <faultHandlers> element which contains a <catch> sub-elements and where the <catchAll> sub-element is used.

Event handling:

The next step in the translation of a BPEL4WS process using the TM is the translation of the process' event handlers.
Event handlers in a BPEL4WS (sub-)process are grouped by the <eventHandlers> element. This element can contain two kinds of events:

- Alarm events that are defined by an <onAlarm> element. The alarm event is triggered when a time-out occurs. A time-out for an <onAlarm> element occurs when the duration expression or deadline expression of the <onAlarm> element results to "*true*".
- Events that are triggered when a certain message arrives during the process instance's execution, which are defined by an <onMessage> element.

The event handlers for a process instance are activated by a *start activity*. The activation initiates all timers that raise an alarm event and the waiting for incoming messages.

Figure 2.12 below displays the translation of an <eventHandlers> element in which one alarm event and one incoming message event can occur. The BPEL4WS semantics define that an alarm event can occur only once during the lifetime of a process instance, that is why the token is consumed by a collector when the alarm event has been handled. The events that are triggered when a certain message arrives may occur more than once during the lifetime of a process instance.

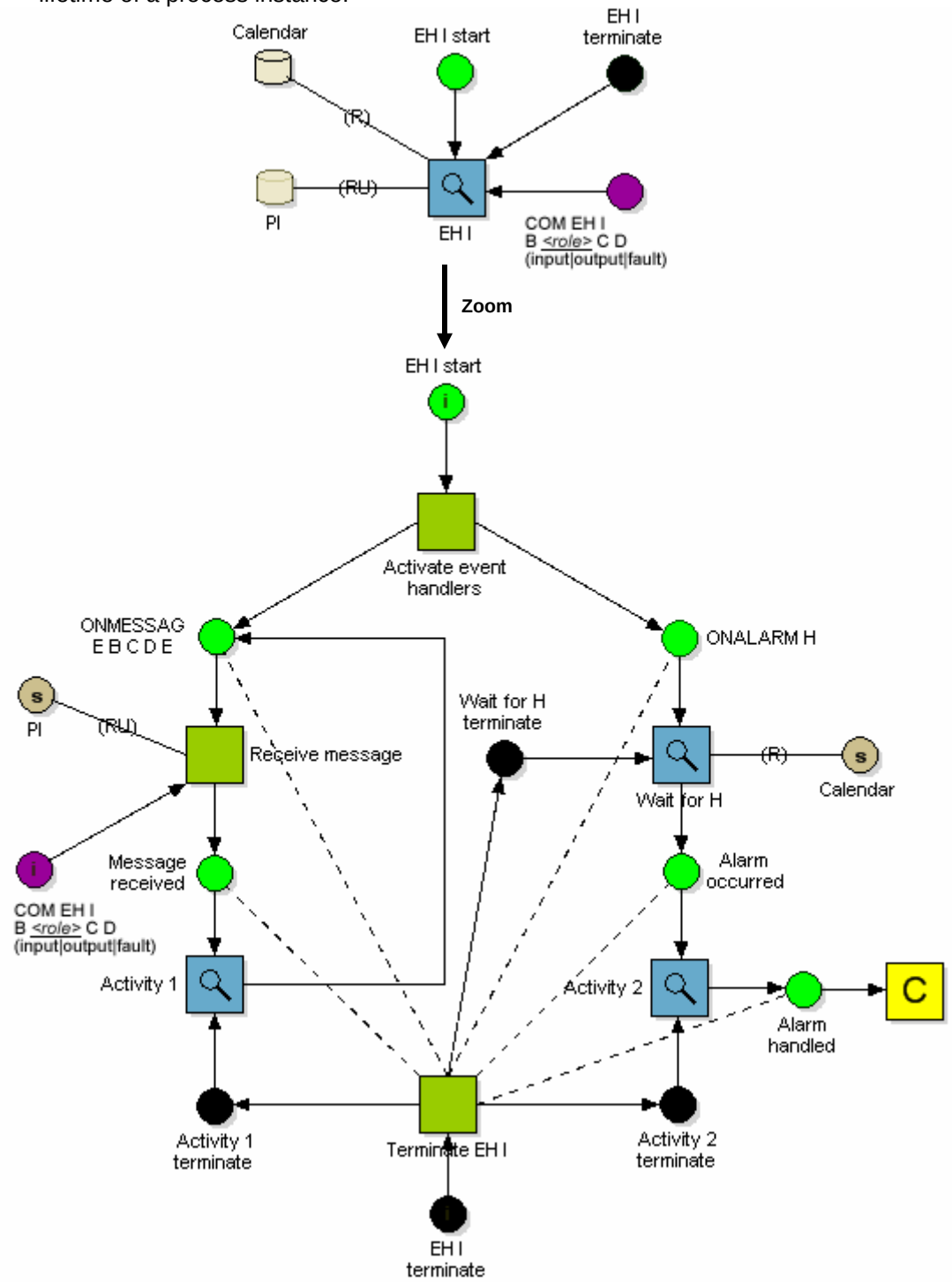


Figure 2.12: The <eventHandlers> element with one possible alarm event and one incoming message event.

Compensation handling:

The next step in the translation of a BPEL4WS process using the TM is the translation of the process' compensation handler. A compensation handler in BPEL4WS is defined by the `<compensationHandler>` element and can be used to define activities that perform "undo"-actions

Therefore the `<compensationHandler>` element is nothing more than a wrapper for reversal activities that may only use a snapshot of the status of the process instance when the scope of the compensation handler completed. The translation of a compensation handler can be seen in figure 2.13 below.

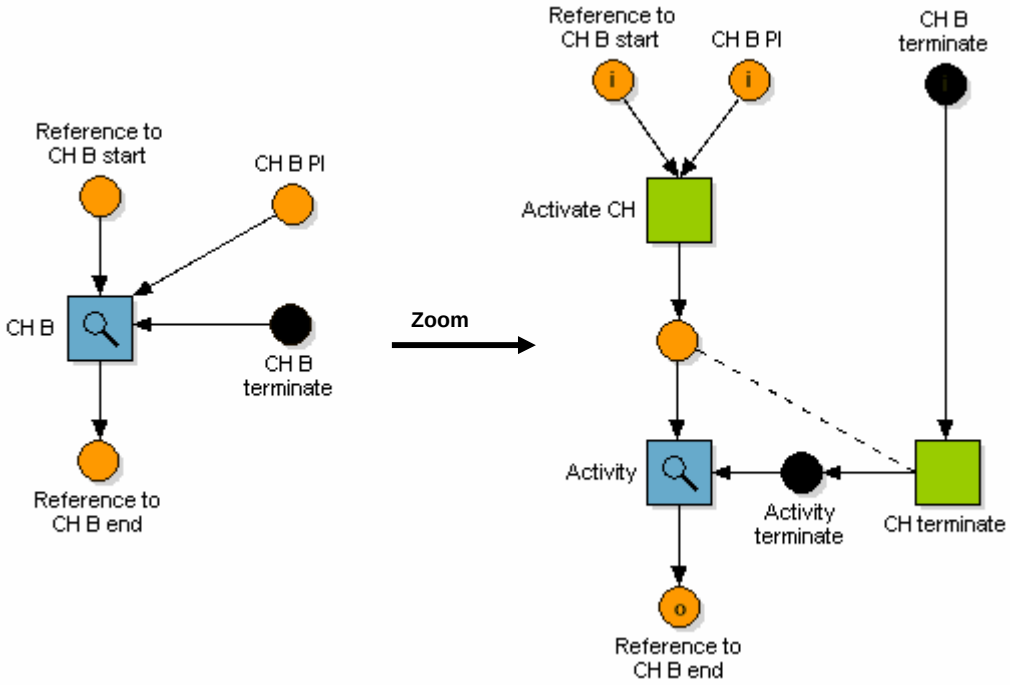


Figure 2.13: The translation of `<compensationHandler>` element of a scope "B".

Note: In the translation above the compensation handler can also be terminated. This can only happen when scope "B" does not equal the scope of the `<process>`.

Start activities:

A BPEL4WS process must be initiated by a *start activity*. A *start activity* creates a new process instance of the process and activates the event handlers of the process. The BPEL4WS specification [BPEL4WS, 2003] also describes that no other basic activities may precede a *start activity* than similarly annotated `<receive>` activities.

In my opinion this logically has the following consequence: it may be preceded by any other structured activity. In case one of the structured activities `<switch>` or `<while>` precedes a *start activity*, problems might occur because of the use of non-initialized variables (this is not mentioned in the BPEL4WS specification).

The BPEL4WS process offers us the following *start activities*:

- **`<receive>`** activity : This basic BPEL4WS activity element describes the waiting for a certain message to arrive (defined by its attributes).
- **`<pick>`** activity : This structured BPEL4WS activity element describes the waiting for several possible message to arrive (each defined by an `<onMessage>` sub-element).

Both BPEL4WS activity elements have an attribute called `createInstance`. By default this attribute has the value "*no*", but for a *start activity* this attribute must have the value "*yes*".

In figure 2.14 below, the translation of a <receive> activity element is displayed which serves as a *start activity* for the process. In this case the "previous" place of the <receive> activity element is a reference place to the place "Process I active", hence the odd arc between the "previous" place and the hierarchical transition, which indicates that the hierarchical transition consumes a token from that place and produces a token after firing in the same place (the arc often is called a *biflow*).

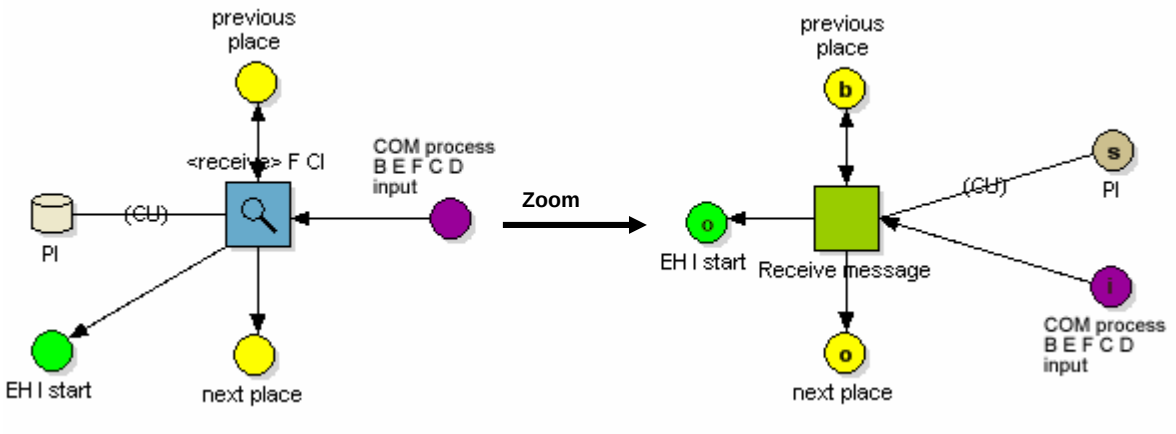


Figure 2.14: The "start activity": <receive>.

In figure 2.15 the translation of a <pick> activity with two possible incoming messages is displayed. A <pick> activity may not contain any alarm events when it is a *start activity*. Also notice that for each incoming message an activity must be defined (the <pick> activity element is a structured BPEL4WS activity element).

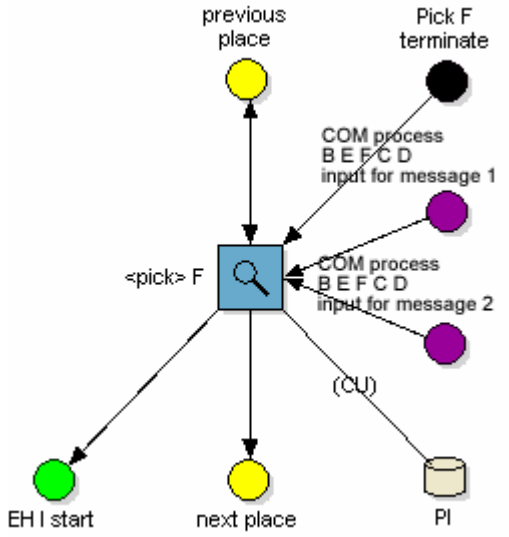


Figure 2.15: The "start activity": <pick>.

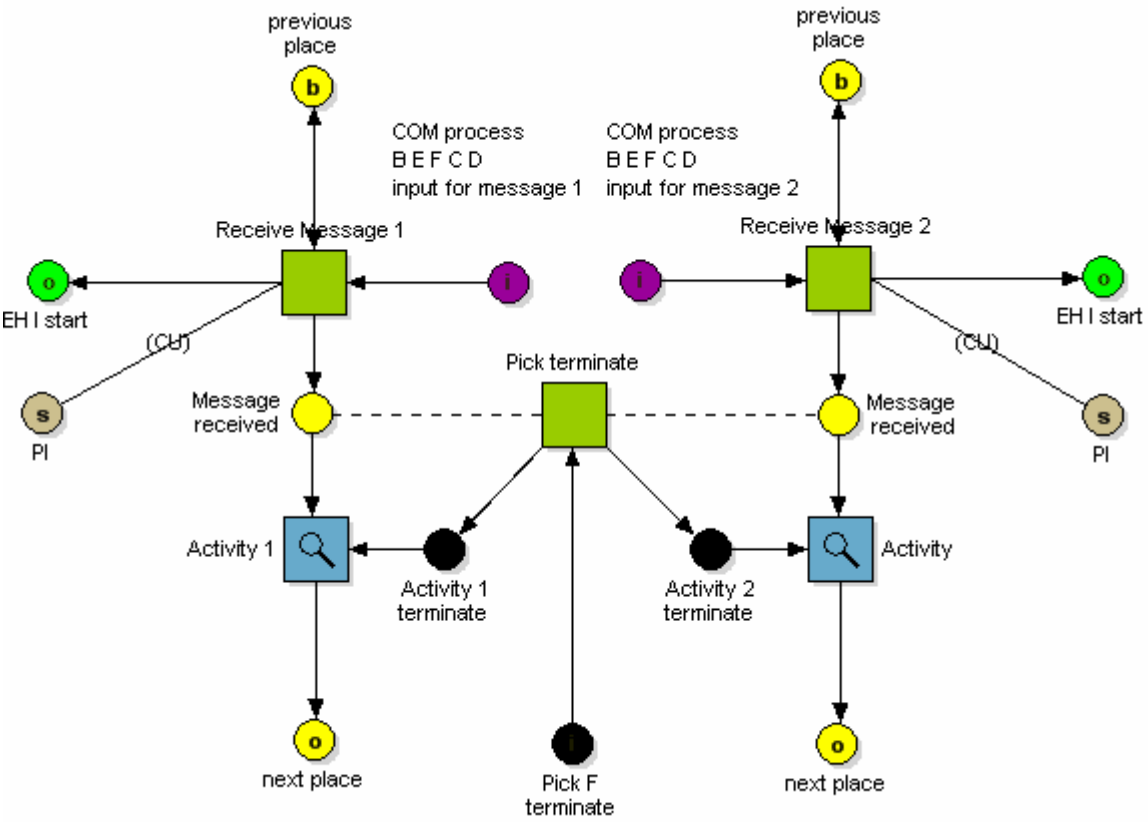


Figure 2.15 (continued; zooming in on the hierarchical transition): The "start activity": <pick>.

Routing of activities:

As described earlier (see Chapter 1), three types of routing in HLPN are used in this GA: sequential routing, parallel routing and selective routing. BPEL4WS has special activity elements for each of these types, namely:

- **<sequence>** activity element : Represents sequential routing.
- **<flow>** activity element : Represents parallel routing.
- **<switch>** activity element : Represents selective routing.

In figure 2.16 below a <sequence> activity element is used to indicate that the two activities should be executed in a sequence. This means that activity 1 is executed before activity 2.

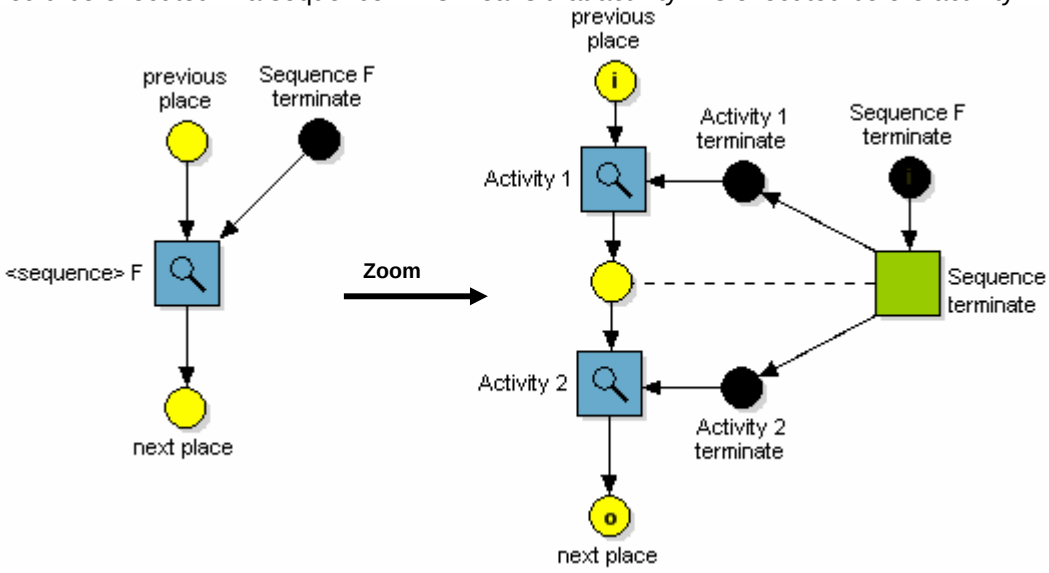


Figure 2.16: The <sequence> activity element with two activities.

In figure 2.17 below a <flow> activity element is used to indicate that the two activities should be executed concurrently. If another activity would be added to the same <flow> activity, then another branch will be created in the hierarchical transition of the <flow> activity element in the same way as is done for the other activities. Also notice that the "previous" place and the "next" place of each activity should refer to respectively the start place and the end place of the branch the activity belongs to.

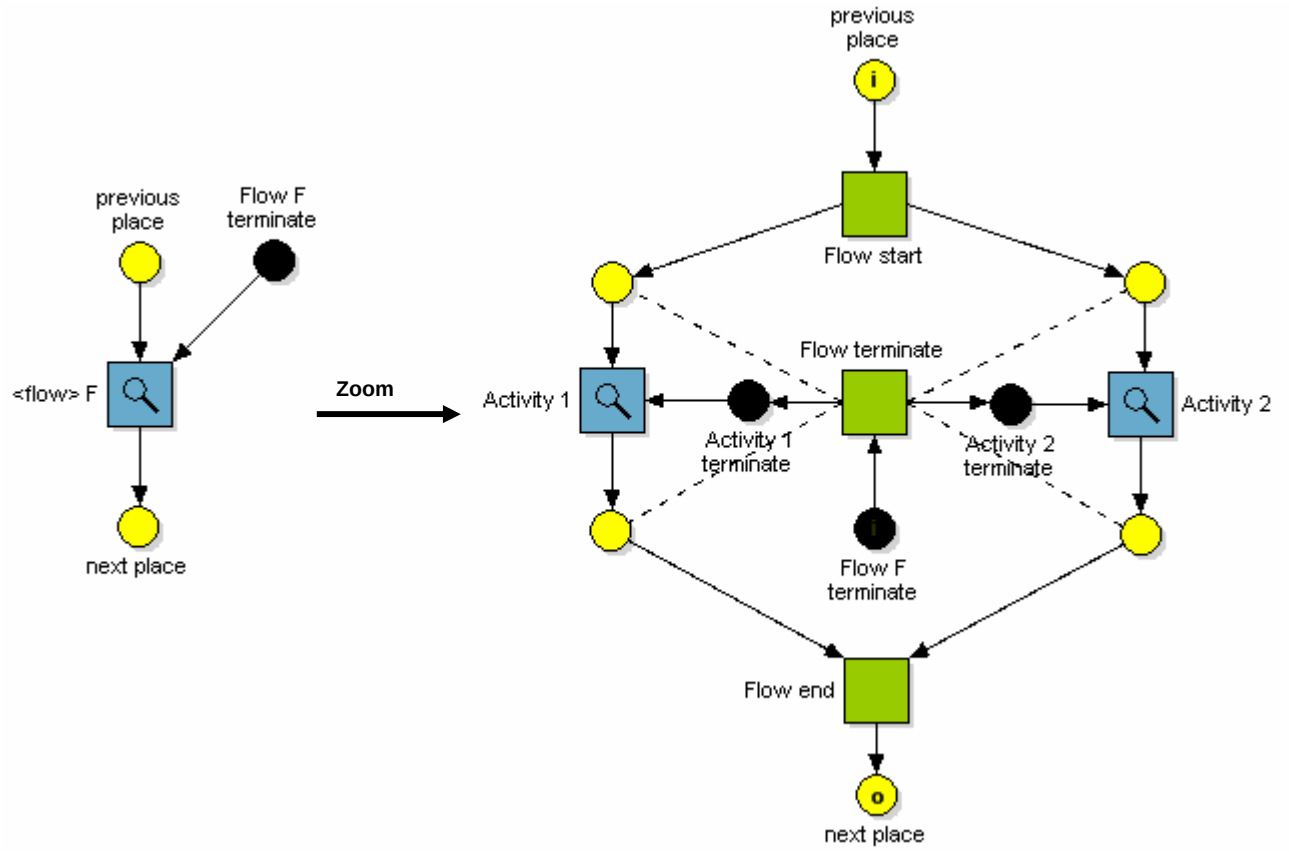


Figure 2.17: The <flow> activity element with two activities.

The <switch> activity element needs some more attention. In the <switch> activity element, one can define several <case>s. Each <case> describes a path in the selective routing. A <case> contains a condition that has to be met before the process can continue in the direction of the <case>. In the situation where none of the <case>'s conditions can be met, the process cannot continue. In that case, one can define a default "case" called <otherwise>. If one omits the <otherwise> element, then BPEL4WS introduces a "stealth" default "case".

Figure 2.18 displays the graphical representation of a <switch> activity element using one <case> element and one <otherwise> element. The <switch> activity element first checks all cases in the XOR transition "Case?". In the graphical representation below the condition of a <case> element is indicated with the parameter **B**. This condition must be a Boolean condition.

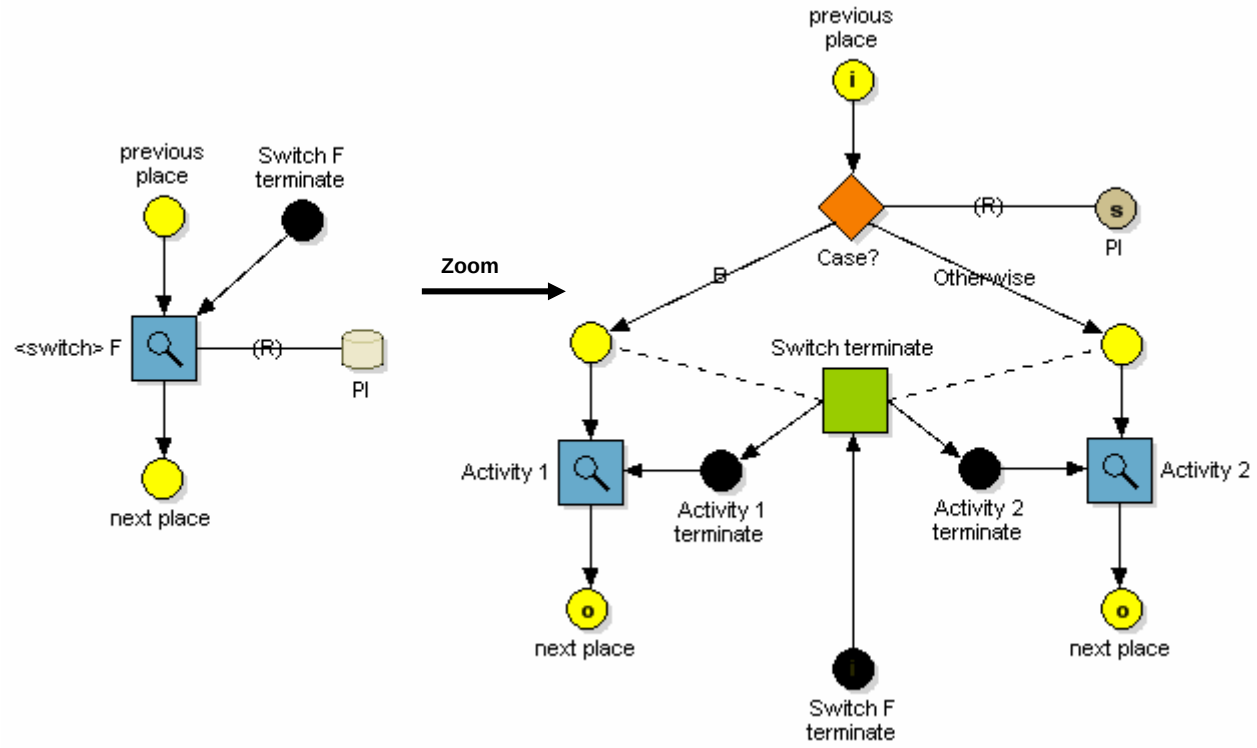


Figure 2.18: The <switch> activity element with one <case> B and one <otherwise>.

Figure 2.19 displays the graphical representation of the same <switch> activity element as is used in figure 2.18 above except for the fact that in this case the <otherwise> element is omitted, which means that BPEL4WS adds a "default" case to the <switch> activity element as can be seen in the figure below.

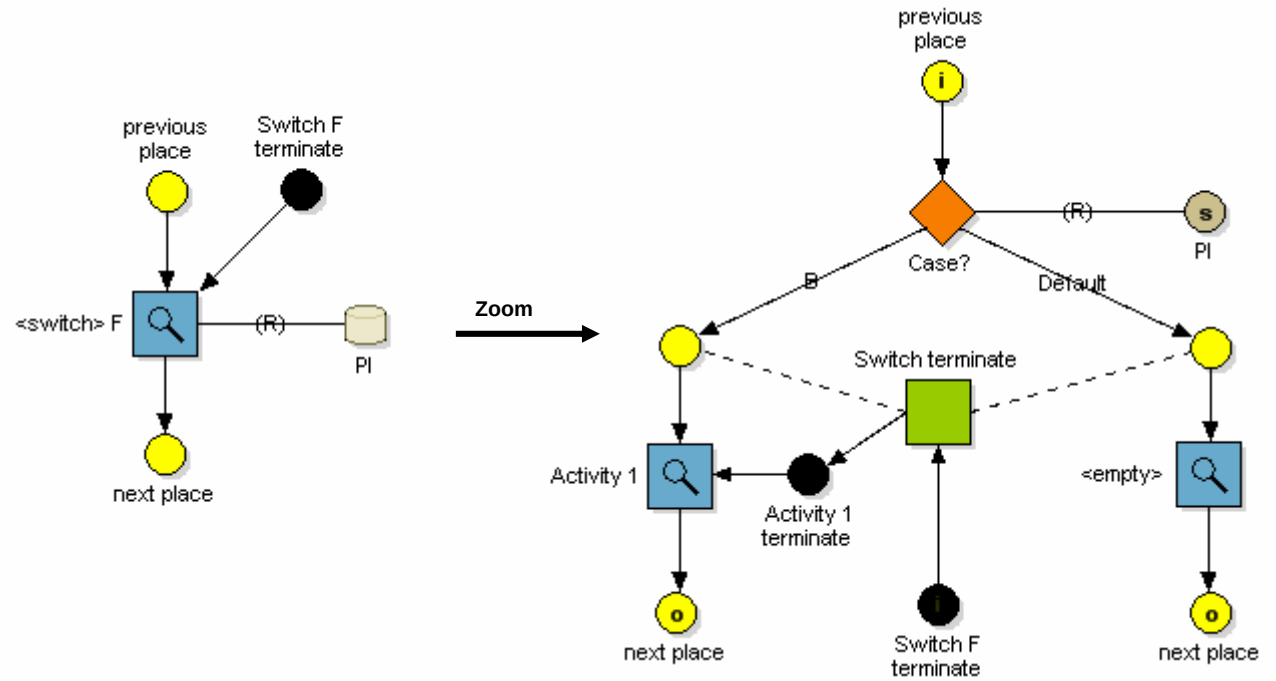


Figure 2.19: The <switch> activity element with one <case> B and where the <otherwise> element has been omitted.

Coupling the activities concerning the termination / halting of the process:

Once all activities of the Main activity have been translated using the TM one still needs to check some couplings concerning the termination / *halting* of the process.

In principle the **black** places and transitions connected to them do not necessarily have to be present when there is no `<terminate>` activity element defined in the process, but if somewhere in the process this activity element is used, then every BPEL4WS (activity) element must be provided with the correct termination of the particular BPEL4WS (activity) element. In general, a place is added to the basis construction for the translation of a BPEL4WS (activity) element, which should initiate the termination of the BPEL4WS (activity) element when it receives a token. Inside the hierarchical transition of the BPEL4WS (activity) element, one needs to add a transition that is connected to the reference place of the termination place, which immediately kills the BPEL4WS (activity) element. It kills the BPEL4WS (activity) element by connecting all places inside the same hierarchical transition via a reset arc-and producing a token inside the termination places of the hierarchical transitions, e.g. as can be seen in figure 2.19 above.

One can terminate a process by using the `<terminate>` activity element. This activity element should produce a token in the reference place of the place “Process terminate” in the hierarchical transition it belongs to. This means that once a `<terminate>` activity element is used, one must add a reference place to the place “Process terminate” for the scope the activity element belongs to and each parent scope of that scope. Figure 2.20 below displays the termination of a certain scope B.

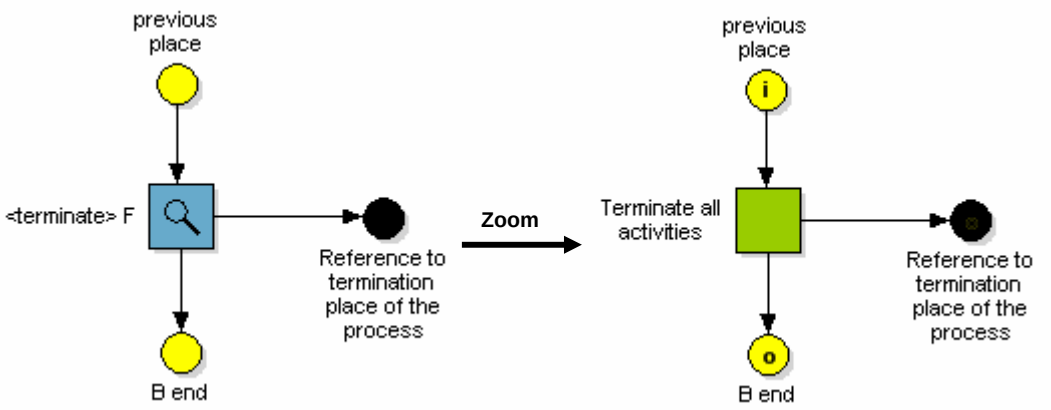


Figure 2.20: Termination of the process from a scope “B”.

Note: The termination of a process can only be implemented for processes that do not represent an abstract process as is defined in [BPEL4WS, 2003].

If a (sub-)process needs to be halted, then the Main activity of that (sub-)process must halt all its activities as soon as possible. This means that when a fault occurs, the fault handler will initiate the halting of the (sub-)process’ Main activity. In general a place is added to the basis construction for the translation of a BPEL4WS activity element, which should halt the activities inside the hierarchical transition of the BPEL4WS activity element (This does not count for the basic BPEL4WS activity elements).

2.2.2. Example: Initial example of [BPEL4WS, 2003]

In this subsection the "Initial example of the BPEL4WS specification ([BPEL4WS, 2003], subsection 6.1) is translated using the TM. The results of the translation are presented in their graphical notation.

Step 1: Take the BPEL4WS file of the process one wants to translate

The source of this file can be found in the BPEL4WS specification on page 16-23 and in Appendix G.2.

Step 2: Retrieve the associated WSDL files that are referred to in the BPEL4WS file

The source of this file can be found in the BPEL4WS specification on page 16-23 and in Appendix G.1.

Step 3: Check if the elements are in conformance with the BPEL4WS specification

Since it is an example of the BPEL4WS specification, the elements are presumed to be in conformance with the BPEL4WS specification.

Step 4: Start the translation by creating a new EPNML file

There is no graphical representation available for the creation of a new EPNML file.

Step 5: Translate the `<process>` element

The `<process>` element of the initial example is shown below in code 2.7.

```
<process name="purchaseOrderProcess"
       targetNamespace="http://acme.com/ws-bp/purchase"
       xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
       xmlns:lns="http://manufacturing.org/wsdl/purchase">
```

Code 2.7: Definition of the `<process>` element in the initial example.

In the `<process>` element above the default settings for a BPEL4WS process are taken, which means that the attributes that are omitted in the definition above have the following values:

- *queryLanguage* : http://www.w3.org/TR/1999/REC-xpath-19991116
- *expressionLanguage* : http://www.w3.org/TR/1999/REC-xpath-19991116
- *suppressJoinFailure* : "no"
- *enableInstanceCompensation* : "no"
- *abstractProcess* : "no"

The graphical representation of this `<process>` element is shown in figure 2.21 below.

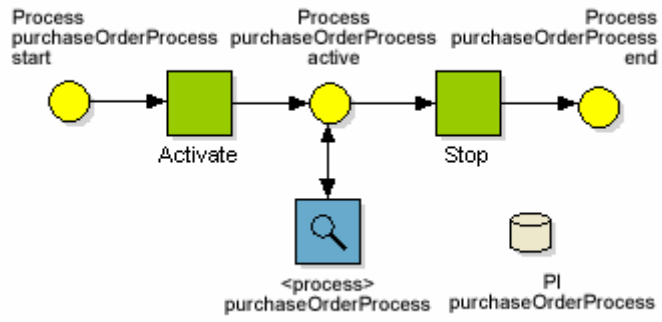


Figure 2.21: Translation of the `<process>` element of the initial example.

Step 6: Translate all communication ports

In the initial example there are four partner links defined, which are shown in code 2.8 below.

```
<partnerLinks>
	<partnerLink name="purchasing" partnerLinkType="lns:purchasingLT"
		myRole="purchaseService"/>
	<partnerLink name="invoicing" partnerLinkType="lns:invoicingLT"
		myRole="invoiceRequester" partnerRole="invoiceService"/>
	<partnerLink name="shipping" partnerLinkType="lns:shippingLT"
		myRole="shippingRequester" partnerRole="shippingService"/>
	<partnerLink name="scheduling" partnerLinkType="lns:schedulingLT"
		partnerRole="schedulingService"/>
</partnerLinks>
```

Code 2.8: Definition of the <`partnerLinks`> element in the initial example.

Here I only translate the first partner link definition, the other partner link definitions can be translated in a similar way.

The first partner link definition has the partner link type definition "`lns:purchasingLT`", which can be found in the associated WSDL file indicated by the namespace prefix `lns`. The partner link type is defined as shown in code 2.9 below.

```
<plnk:partnerLinkType name="purchasingLT">
	<plnk:role name="purchaseService">
		<plnk:portType name="pos:purchaseOrderPT"/>
	</plnk:role>
</plnk:partnerLinkType>
```

Code 2.9: Definition of the <`partnerLinkType`> element "purchasingLT".

This partner link type definition has only one role, "`purchaseService`", which happens to be the same role as defined for the *myRole* attribute of the partner link definition. This means that the port type associated with the role belongs to the process "`purchaseOrderProcess`" itself.
The port type definition of the port type "`pos:purchaseOrderPT`" is shown in code 2.10 below, where the namespace prefix `pos` refers to the same WSDL file.

```
<portType name="purchaseOrderPT">
	<operation name="sendPurchaseOrder">
		<input message="pos:POMessage"/>
		<output message="pos:InvMessage"/>
		<fault name="cannotCompleteOrder"
			message="pos:orderFaultType"/>
	</operation>
</portType>
```

Code 2.10: Definition of the <`portType`> element "purchaseOrderPT".

The next step in the translation is the translation of each operation of the port type definition. In this case only one operation exists, "`sendPurchaseOrder`". The operation defines a "Request-Response" interaction pattern in which one fault definition exists.

The graphical representation of this partner link is shown in figure 2.22 below in which I also zoom in on the process itself.

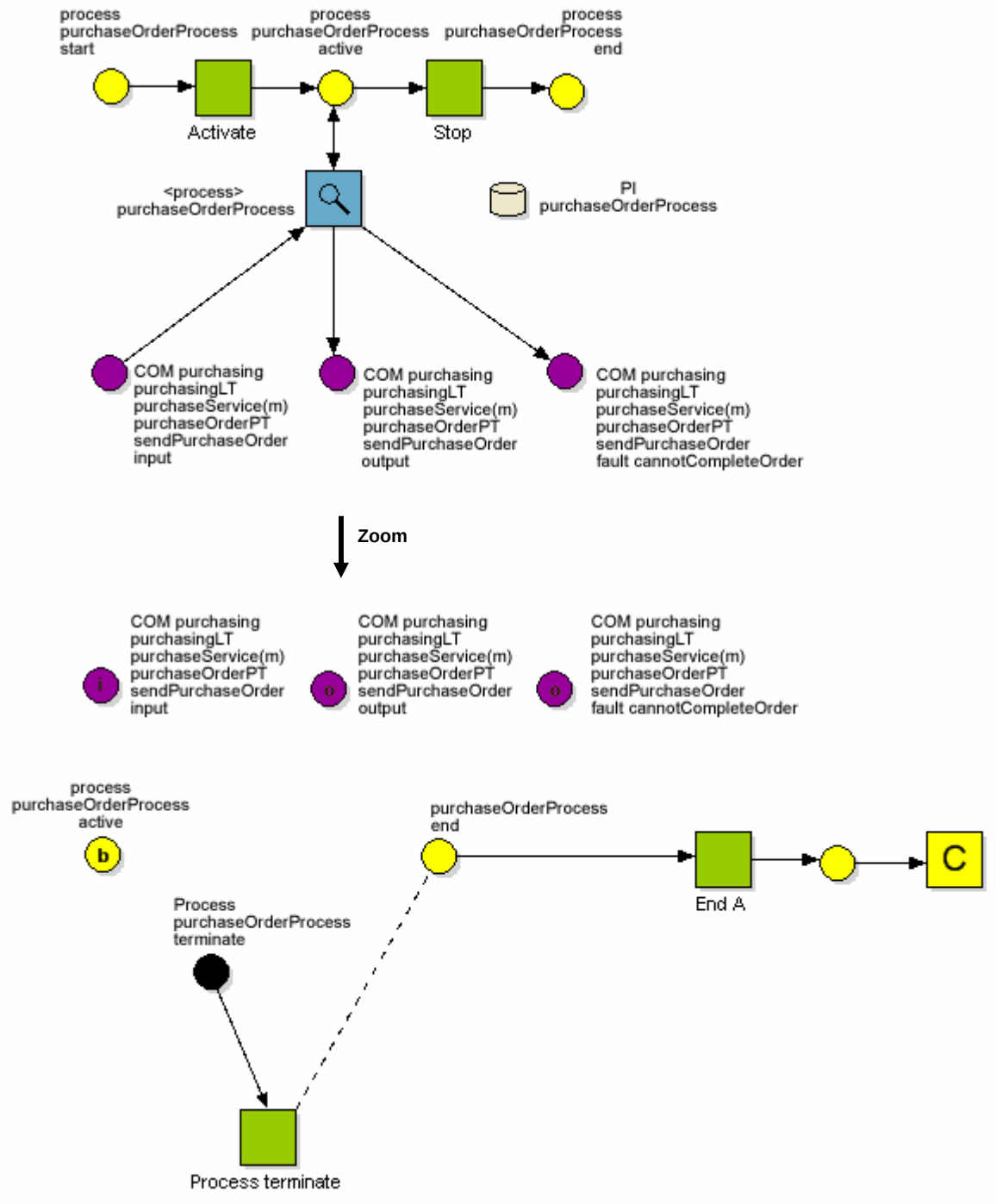


Figure 2.22: Translation of the <partnerLink> element "purchasing" of the initial example.

Step 7: Create the process-token type, message-token types and fault-token types

There is no graphical representation available for the creation of the process-token type, message-token types and fault-token types.

Step 8: Translate the fault handlers

In the initial example there are no sub-processes used (no presence of <scope> elements). This means that there is only one scope which can have a set of fault handlers defined for it and that is the scope belonging to the process (<process> element) itself.

There is only one fault handler defined for the scope and is defined as shown in code 2.11 below.

```
<faultHandlers>
      <catch faultName="lns:cannotCompleteOrder" faultVariable="POFault">
            <reply partnerLink="purchasing" portType="lns:purchaseOrderPT"
                  operation="sendPurchaseOrder" variable="POFault"
                  faultName="cannotCompleteOrder"/>
      </catch>
</faultHandlers>
```

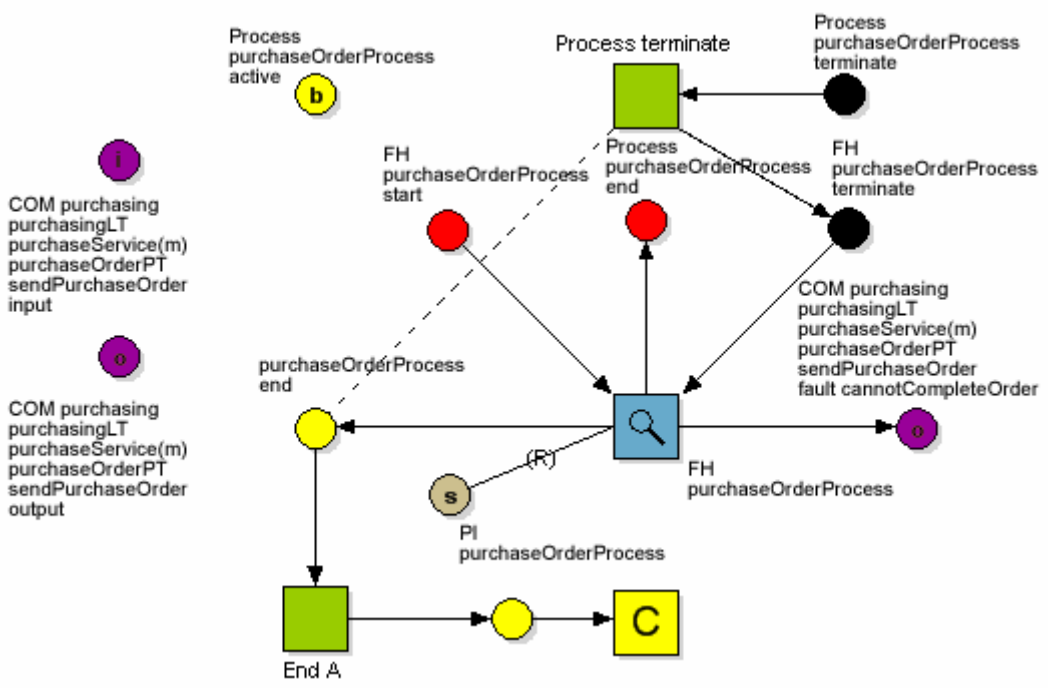
Code 2.11: Definition of the <faultHandlers> element for the scope of the <process> element.

As one can see the "<catchAll>" sub-element is omitted in the definition. This means that BPEL4WS adds a "stealth" default fault handler to the set of fault handlers.

The fault handler that is defined catches a fault named "cannotCompleteOrder" and some values that are provided by the variable "POFault". When this particular fault occurs during the execution of the process, then the activity belonging to this fault handler is executed.

The activity that is defined for this fault handler, a <reply> activity element, sends a message containing the values of the "POFault" variable to the COM-port of the partner link "purchasing" which can handle the fault "cannotCompleteOrder".

The graphical representation of this <faultHandlers> definition in the process is shown in figure 2.23 below.



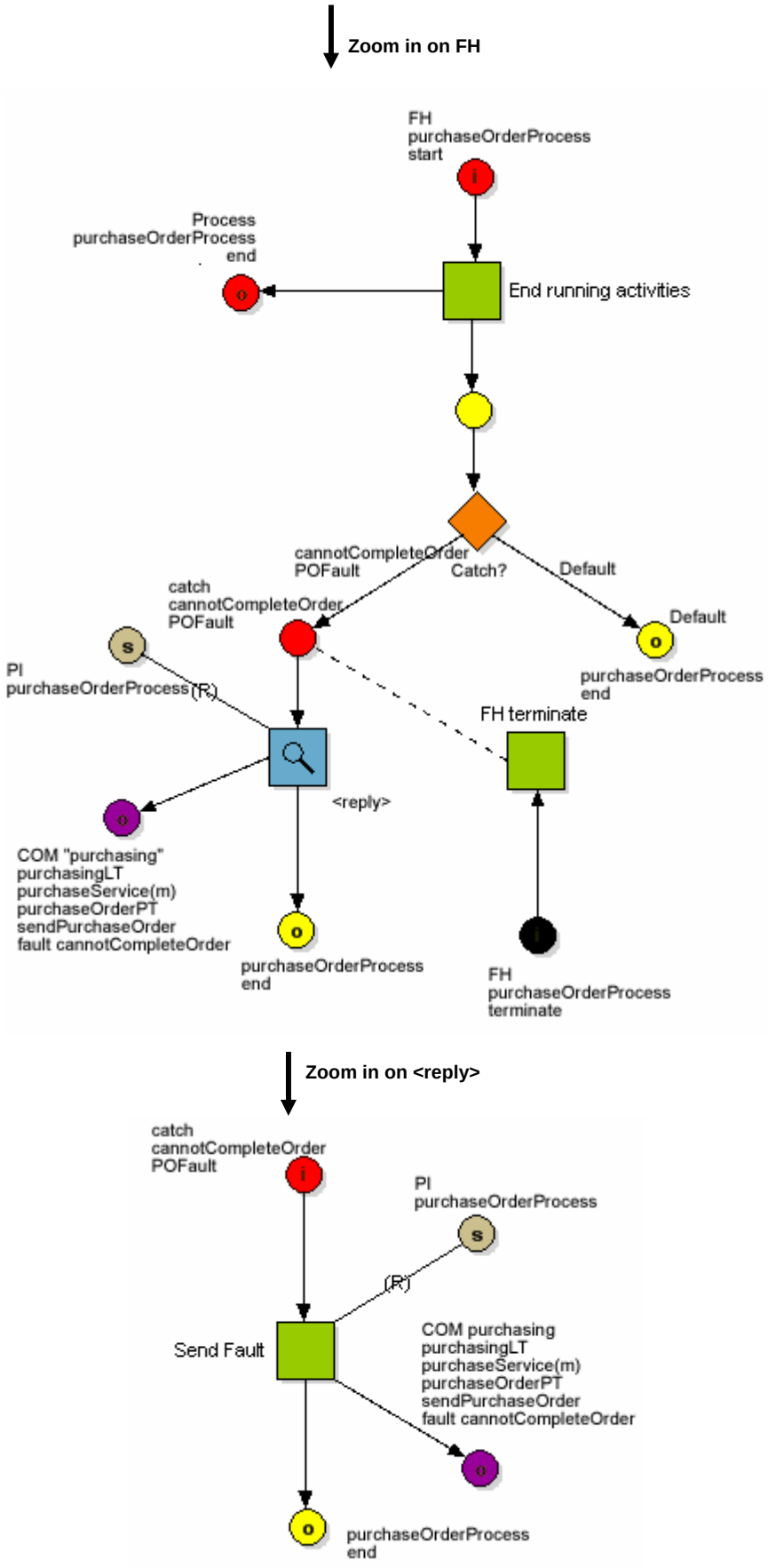


Figure 2.23: Translation of the <faultHandlers> element of the initial example.

Note: The extra check in the <faultHandlers> element's translation for the scope has been left out of the translation, because the <faultHandlers> element belongs to the scope of the process itself.

Step 9: Translate the event handlers

There are no event handlers defined for this example, this also means that the incoming messages do not need to be distributed for each event that uses the incoming message and the process itself.

Step 10: Translate the compensation handler

There is no compensation handler defined for this example.

Step 11: Translate the main process

In the main process the following happens (for translations see G.3):

1. Receive a message via the COM-port of partner link "purchasing" and storing it in the variable "PO".
2. Concurrent execution of three sequences, with control flow:
    a. First sequence:
        i. Copying the part "customerInfo" of the variable "PO" to a part "customerInfo" in the variable "shippingRequest".
        ii. Invoking a "Request-Response" interaction "requestShipping" with a process that belongs to the partner link "shipping". The request holds the same value as the variable "shippingRequest" and the response is stored in the variable "shippingInfo". If a fault occurs in this interaction process, then it is thrown to the fault handlers of the process. Also this activity enables the link "ship-to-invoice".
        iii. Waiting for a message to be received via de COM-port of partner link "shipping" with operation "sendSchedule" and the message is stored in the variable "shippingSchedule". Also this activity enables the link "ship-to-scheduling".
    b. Second sequence:
        i. Invoking a "One-way" interaction "initiatePriceCalculation" with a process that belongs to the partner link "invoicing". The request holds the same value as the variable "PO".
        ii. Invoking a "One-way" interaction "sendShippingPrice" with a process that belongs to the partner link "invoicing". The request holds the same value as the variable "shippingInfo". This activity can only be executed if the link "ship-to-invoice" is enabled.
        iii. Waiting for a message to be received via de COM-port of partner link "invoicing" with operation "sendInvoice" and the message is stored in the variable "Invoice".
    c. Third sequence:
        i. Invoking a "One-way" interaction "requestProductionScheduling" with a process that belongs to the partner link "scheduling". The request holds the same value as the variable "PO".
        ii. Invoking a "One-way" interaction "sendShippingSchedule" with a process that belongs to the partner link "scheduling". The request holds the same value as the variable "shippingSchedule". This activity can only be executed if the link "ship-to-scheduling" is enabled.
3. Sending a response with the contents of the variable "Invoice" via the COM-port of partner link "purchasing".

Step 12: Coupling the activities concerning the termination / halting of the process

In this process model, all activity termination places need to be coupled to the termination transition of the scope they are defined in.
This also counts for the activity ending places (used by the fault handler to stop all activities in the main process).

The result of the total translation can be seen in Appendix G.3.

2.3. Mapping Model

2.3.1. Creating the Mapping Model

The TM provides us with a set of predefined handles for each BPEL4WS element. The Mapping Model (hereafter abbreviated as *MM*) is a model that uses this predefined set of handles to offer the developer of BPEL4WS processes a guide how to model a BPEL4WS process with these handles.

First the MM had the same form as the TM, but I realized that I was redefining every element again and the logical question then would be: “Why did you create the TM?” This is not what I wanted and I chose another angle in which a developer can use the handles to develop a graphical representation of the model along with the BPEL4WS representation of the model. The EPNML representation then can be retrieved from the TM. This angle made the TM more important than the first angle I took.

Each section in the MM consists out of four main parts:

- A flow describing which path to follow when creating a BPEL4WS process model using the handles of the TM.
- Indications how the graphical representation of the elements should look in the model of the BPEL4WS process.
- Description of a set of involved parameters that the developer can use to give the element its actual form.
- Description of the BPEL4WS element representation using the set of involved parameters.

In fact each section in the MM describes a sort of “wizard” for a particular activity in a BPEL4WS process. The following “wizards” can be identified:

- Creating the outline for a new process.
- Creating the COM-ports for a process.
- Creating the temporary storage possibilities (variables).
- Creating the correlation sets that are used in the message correlation.
- Creating the fault handlers for a particular scope.
- Creating the compensation handler for a particular scope.
- Creating the event handlers for a particular scope.
- Creating the activity that has to be performed:
    - Receiving a message.
    - Sending a message.
    - Routing of activities.
    - Creation of a sub-process.
    - Iterating an activity.
    - Assignments to variables.
    - Terminating the process.
    - Asking for compensation of a particular scope in the process.
    - Throwing faults.
    - Waiting.
    - Doing nothing.
- Creating the control flow in an activity.

The MM can be found in Appendix F of this thesis.

2.3.2. Example: Initial example of [BPEL4WS, 2003]

In subsection 2.2.2 the initial example of the BPEL4WS specification [BPEL4WS, 2003] has been translated using the TM. In this subsection the MM is used to create the same model using the handles.

The details about the process can be found in subsection 2.2.2 and are therefore not described here.

The MM has to perform the following steps in the creation of this model (A "Request-Response" process):

1. Create the beginning of the process.
2. Insert sequential routing for the activities of the main process.
3. Insert the handle "receive" without instance creation (since it is the first activity it also needs to put a token back in the place "Process purchaseOrderProcess active").
4. Insert parallel routing for activities inside the scope of the sequential routing.
    a. Add the controls (=links) to the process instance.
    b. Insert three sequential routings for activities inside the scope of the parallel routing:
        i. First sequential routing:
            1. Insert an "assignments" handle with one <copy>-action.
            2. Insert an "invoke" handle with "Request-Response" interaction pattern. Also enable the link "`ship-to-invoice`".
            3. Insert a "receive" handle without instance creation. Also enable the link "`ship-to-scheduling`".
        ii. Second sequential routing:
            1. Insert an "invoke" handle with "One-way" interaction pattern.
            2. Insert an "invoke" handle with "One-way" interaction pattern, which can only be executed when the link "`ship-to-invoice`" is enabled.
            3. Insert a "receive" handle without instance creation.
        iii. Third sequential routing:
            1. Insert an "invoke" handle with "One-way" interaction pattern.
            2. Insert an "invoke" handle with "One-way" interaction pattern, which can only be executed when the link "`ship-to-scheduling`" is enabled.
5. Insert the handle "reply response" handle.
6. Create fault handlers for the process scope:
    a. Insert a normal "fault handlers" handle with "default" case.
    b. Add a catch to the fault handlers for the process scope.

In the rest of this subsection each step described above is described in detail.

Step 1: Create begin of the process

In this step the "empty" process is created. In this step one also defines if the process has platform-specific means for compensation of the whole process (attribute `enableInstanceCompensation` of BPEL4WS). In this case we do not want this kind of compensation; thus we need to insert the "Process" handle 1 in our model. Since our model is called "`purchaseOrderProcess`", the graphical representation of this handle is the same as in figure 2.21.

When one looks at the involved parameters, one notices that these parameters represent the settings of a BPEL4WS process and the settings part of the "process" token type definition. One only needs to set the name of the process (parameter **A**), everything else has its default value. In code 2.12 below the "process" token type definition is show and in code 2.13 the BPEL4WS representation of the handle.

```
<token type="process" id="purchaseOrderProcess">
      <name> purchaseOrderProcess </name>
      <location>location of the process</location>
      <!-- <namespaces> Can be added here -->
      <!-- <variables> Can be added here -->
      <!-- <correlationSets> Can be added here -->
      <!-- <links> Can be added here -->
</token>
```

Code 2.12: Definition of the <token> element for process "purchaseOrderProcess" in the EPNML file.

```
<process name="purchaseOrderProcess"
      targetNamespace=" location of the process"
      <!-- <namespaces> Can be added here -->>

      <!-- <variables> Can be added here -->
      <!-- <correlationSets> Can be added here -->

      <!-- <links> Can be added here inside a <flow> element -->
</process>
```

Code 2.13: Definition of the <process> element for process "purchaseOrderProcess" in the BPEL4WS file.

Step 2: Insert sequential routing

In this "Request-response" process one always needs a sequential routing of the main activities, because this process can only receive one kind of messages and sends only one kind of messages back.
Figure 2.24 displays how the sequential routing can be added to the main activity of the process by zooming in on the result of the previous step. Steps 3, 4 and 5 of this example describe the activities inside the "sequential routing" handle.

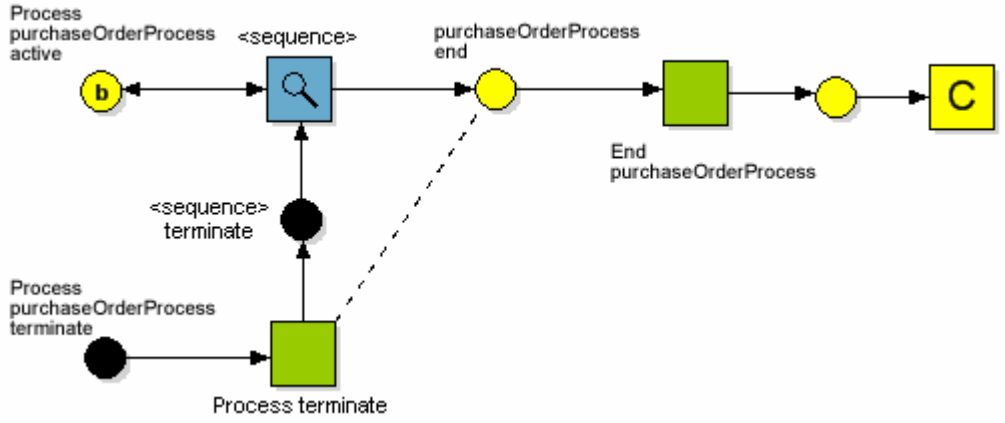


Figure 2.24: Zooming in on the "Process" handle 1 with a sequential routing of its main activity.

Step 3: Insert "receive" handle without instance creation

In a "Request-Response" process one first needs to define an activity for the incoming request message. As described in step 11 of the example in subsection 2.2.2, the "`purchaseOrderProcess`" can receive a request via its COM-port of the partner link "`purchasing`". This is the only request the process can receive at the start of the process. Therefore one needs to add the "receive" handle as the first activity inside the "sequential routing" handle.
Figure 2.25 below shows how this activity can be added inside the "sequential routing" handle including the translation of the COM-port of the partner link "`purchasing`".

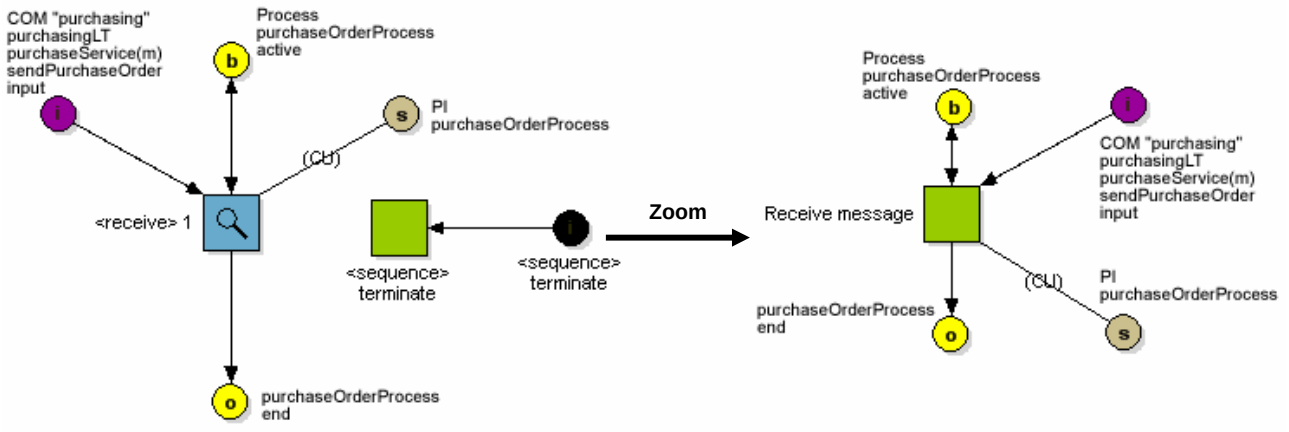


Figure 2.25: The "sequential routing" handle with one "receive" handle.

Step 4: Insert parallel routing

The "`purchaseOrderProcess`" process starts after the receiving of a "PO" three parallel sequences of activities:

1. Communication with a shipping service.
2. Communication with an invoicing service.
3. Communication with a scheduling service.

These sequences and internal activities can be added to the "parallel routing" handle in the same way as the "sequential routing" handle for the "Process" handle 1.

In this step only the adding of the "parallel routing" handle inside the "sequential routing" handle of the main activity is shown (see figure 2.26). The graphical representation of the above described sequences can be found in Appendix G.3.

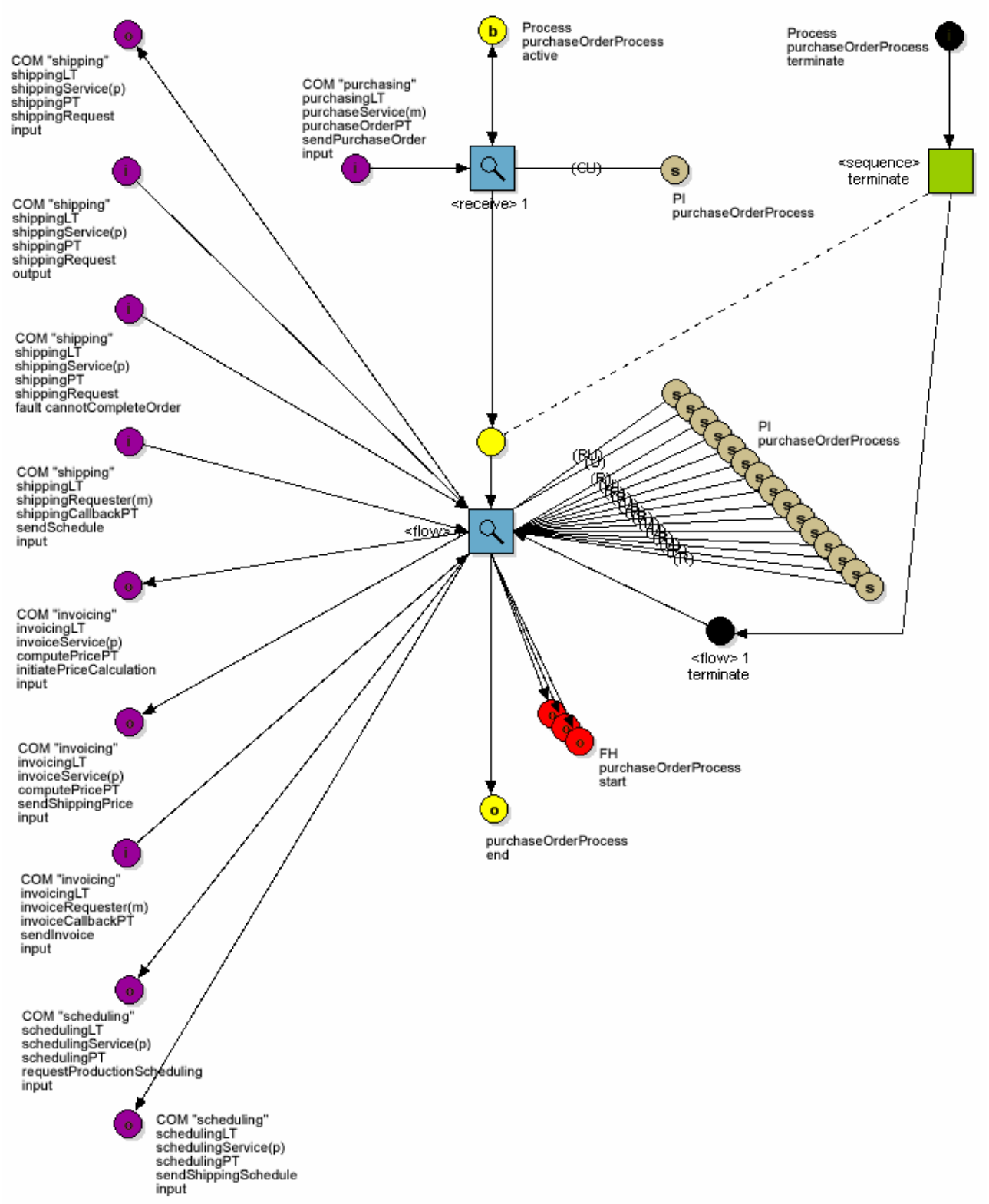


Figure 2.26: One "receive" handle and the "parallel routing" handle inside the "sequential routing" handle.

Note: An intermediate place has been added between the "receive" handle and the "parallel routing" handle. Also termination of the flow needs to be coupled to the termination of the whole process.

Step 5: Insert "reply response" handle

This is the final step of the main activity. In this step a "reply response" handle is inserted inside the scope of the "sequential routing" handle, which causes a response to be send back via the COM-port of the partner link "`purchasing`".
Figure 2.27 below shows how the "reply response" handle is added to the "sequential routing" handle.

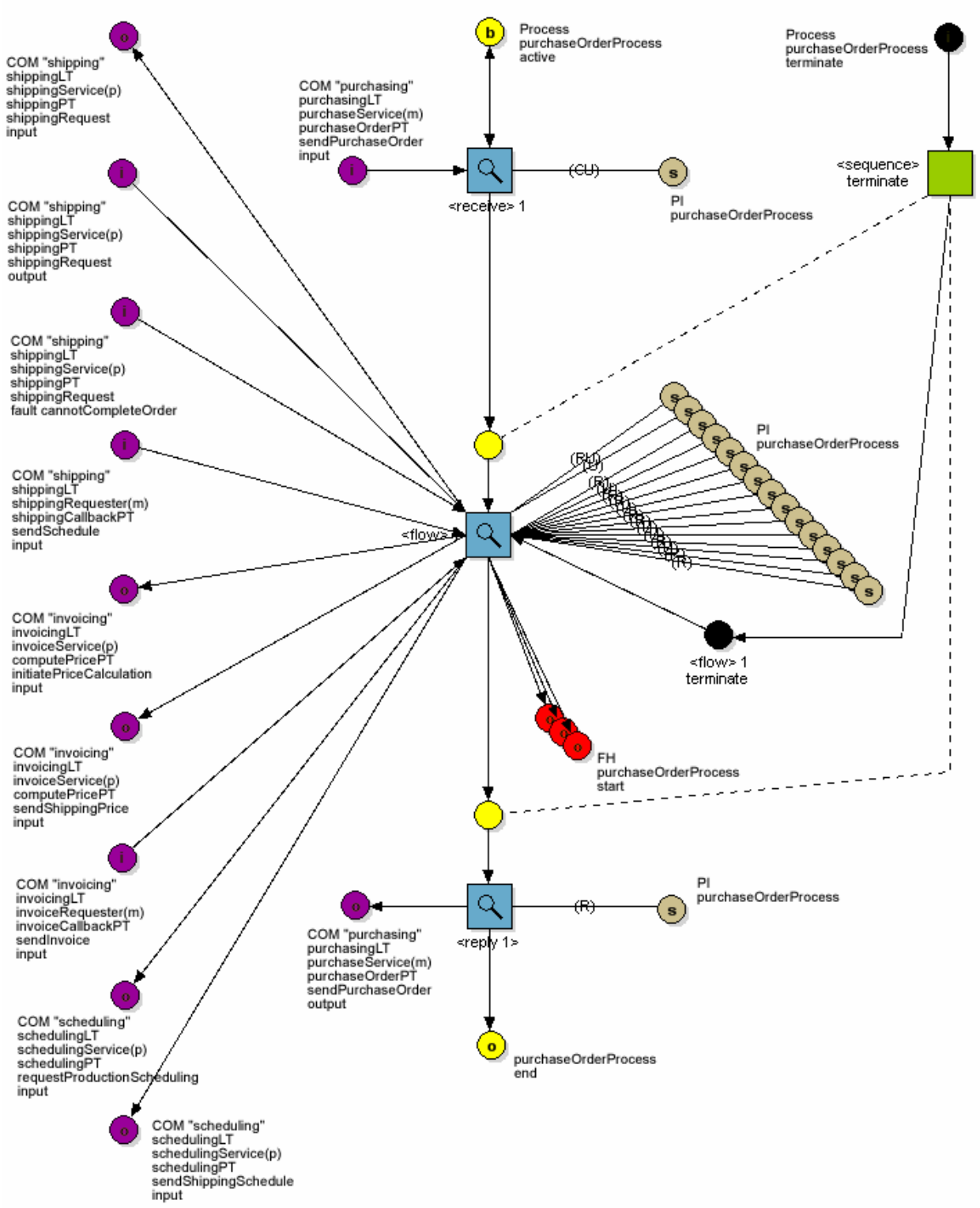


Figure 2.27: The "receive" handle, the "parallel routing" handle and the "reply response" handle inside the "sequential routing" handle.

Step 6: Create fault handlers for the process scope

The final step in the creation of this process is the creation of a fault handler for the whole process. In this process only one fault handler is defined to catch the fault "`cannotCompleteOrder`" along with the variable "`POFault`", which contains the message belonging to the fault. This means that there is no "catch all faults" case present in the fault handlers for the process, which means that one needs to insert a normal "fault handlers" handle with a "default" case.
In figure 2.23 the second and third figure display the normal "fault handlers" handle with a "default" case.

Note: Adding a "fault handlers" handle to a process also calls for the ending of the main activities when a fault occurs. This means that inside the "sequential routing" handle the ending of the running activities must be modeled as described in subsection 2.2.1. The transition that initiates the ending of the activities must be coupled to the place "`Process purchaseOrderProcess end`" of the "fault handlers" handle of the process (as can be seen in figures G.2 and G.5 in Appendix G).

The final result of the translation is the same as the result of the one in the example of subsection 2.2.2 (see Appendix G.3).

2.4. Correctness of the Translation Model & Mapping Model

The result of the TM and MM is a HLPN-based process model of a BPEL4WS process model. The correctness of this result can be validated using tools that are capable to load the result of this model, an EPNML file (using the TM) or a BPEL4WS file with the associated WSDL file(s) (using the MM and TM).

The correctness of my translation lies in the use of compositional modeling and hierarchical structuring of the translations. Each BPEL4WS element (except for the `<partnerLink>`s, `<partner>`s, `<variable>`s, `<correlationSet>`s and `<link>`s) that has been translated in the TM uses the basis construction as described in subsection 2.2.1 "Element translation". Besides the basis construction also the semantics of the specific BPEL4WS element play a role in the correctness of my translation.

The basis construction suggests that one token is consumed from the input place by the hierarchical transition and the hierarchical transition produces one token in its output place. This means that inside the hierarchical transition the working of the BPEL4WS (activity) element must be in conformance with this restriction. In general there are four factors that disturb this restriction, namely:

- Communication.
- Fault handling.
- Termination.
- Compensation handling.

In the case of communication, the hierarchical transition may contain one or more COM-port places that are either used as an input place or output place. This means that the "message" tokens residing in the input places must be consumed by the hierarchical transition and the hierarchical transition may produce "message" tokens in these output places. If tokens are produced in the output places that represent a COM-port, then these tokens must be consumed by a collector if the process is not coupled to another process.

In the case of fault handling there are two difficulties that need attention:

- BPEL4WS activity elements that throw a fault.
- The fault handler of a scope that needs to halt all activities of that scope.

The first difficulty is the BPEL4WS activity element that produces a “fault” token which is sent to the fault handler of the (sub-)process it belongs to. As described in subsection 2.2.1 “Fault handling” there is always a fault handler present for the process itself and each sub-process even when it is not defined in the BPEL4WS file. This fault handler consumes the token that has been produced by the BPEL4WS activity element and handles the fault internally.
The second difficulty is that when the fault handler of a (sub-)process is activated all activities of that (sub-)process must be halted. The fault handler of a (sub-)process tells the (sub-)process to halt its Main activity by producing a token in a place, which indicates that Main activity of the (sub-) process should be halted as soon as possible. The hierarchical transition that represents the Main activity must consume this token and handle the halting of the activities inside it as described in subsection 2.2.1“Coupling of all activities concerning the termination / halting of the process”.

In the case of termination the basis structure is extended with an additional place concerning the correct termination of the specific translation of a BPEL4WS element. When terminating the process instance that is running inside a hierarchical transition of the BPEL4WS element’s translation, a transition must be coupled to this place which immediately has to consume all tokens inside the hierarchical transition as described in subsection 2.2.1 “Coupling of all activities concerning the termination / halting of the process”. A process instance is fully terminated when none of that process instance’s “process” tokens are left in the process model. This automatically means that the process cannot end in a normal way, which is also the case in a BPEL4WS process (see subsection 2.2.1 “Process translation”).
If a BPEL4WS activity element’s translation of a `<terminate>` activity element is used in a (sub-) process then there must be a direct link between the place that indicates that the process should be terminated and the activity itself. Once a token resides that place a sort of “flush” action is performed immediately, flushing out all tokens of a certain process instance as described above.

In the case of compensation handling a BPEL4WS activity element `<compensate>` asks for compensation of a particular scope. This means that the hierarchical transition of the <compensate> activity element’s translation produces a token in the place that indicates the start for that (sub-) process’ compensation. Even when there is no compensation handler defined for the (sub-)process in a BPEL4WS file a default compensation handler is present in my translation as is described by the BPEL4WS semantics for compensation handling. This compensation handler’s translation performs its compensation activities and turns back to the `<compensate>` activity element, which proceeds normally.

In all other cases the basis construction and the compositional modeling defend the correctness of the result.

Chapter 3: The PN-BPEL4WS Tool

3.1. Introduction

Another result of the GA is the PN-BPEL4WS Tool, which is an application that one can use as an editor to create a Web Services related BPM using the handles as defined by the TM and MM. This means that the user of this tool can create, load, modify and generate BPEL4WS process models as well as EPNML process models that are in conformance with the TM and MM.
The PN-BPEL4WS Tool is not a simulation tool, which means that models created in this tool cannot be tested in the tool to see how they work. Simulation of these models has to take place in a tool that is capable of loading either an EPNML file or a BPEL4WS file with the associated WSDL file(s).Two of such tools, YASPER and Oracle BPEL Process Manager, are described in the last subsection of this chapter.

In this chapter the following subjects concerning the tool are described:

- Construction : Describes how the PN-BPEL4WS Tool has been constructed.
- GUI environment : Describes the Graphical User Interface (GUI) environment of the tool.
- Relation with the TM and MM : Describes the relation of the tool with the TM and MM of the GA.
- Simulation : Describes how the output of the tool can be used for simulation in the tools YASPER and Oracle BPEL Process Manager.

Note: A user manual and a technical manual of the tool are provided with the PN-BPEL4WS Tool itself.

3.2. Construction

The PN-BPEL4WS Tool has been created using Borland's JBuilder 2005[15]. The tool consists of the following parts:

- BPEL4WS, WSDL, EPNML and Handle classes.
- BPEL4WS and EPNML parsers.
- BPEL4WS and EPNML creators.
- GUI environment.
- Tool's engine.

In this subsection each of the above mentioned parts are described in short except for the GUI environment part, which is described in the next subsection in more detail.

Note: For a full description of the tool's construction and functionality I refer to its technical manual.

3.2.1. Handle, BPEL4WS, WSDL and EPNML classes

The first step in the creation of the PN-BPEL4WS tool was the creation of a set of classes that could be used to create BPEL4WS, WSDL and EPNML objects inside the tool. This resulted in a one-to-one translation of the BPEL4WS, WSDL and EPNML elements to their Java representation in classes. The attributes and sub-elements of an element are converted as members of the class and methods have been added to each class, which one can use to access these members to create, modify and delete their data.
Each class that describes an element also is provided with a method that can be used to translate an object of this class back to its textual form of the element.

[15] http://www.borland.com/us/products/jbuilder/index.html

The Handle classes are the result of the TM and MM. The Handle classes use the BPEL4WS, WSDL and EPNML classes to create objects inside the class to represent the handle as described in the TM and MM. A Handle object can also contain other Handle objects, e.g. a “Sequential Routing” Handle object can contain one or more “Activity” Handle objects.

The Handle objects are used in the tool to model a WS-related business process. Once one has developed a model of the process, this model (a set of Handle objects) can be given to the BPEL4WS creator or EPNML creator to respectively generate a BPEL4WS file with associated WSDL file(s) or an EPNML file of the (business) process model that is currently active in the tool.

3.2.2. BPEL4WS and EPNML parsers

The second step in the creation of the tool consisted of the creation of the BPEL4WS and EPNML parsers.

The input for the BPEL4WS parser is a BPEL4WS file (file with the extension .bpel) and the associated WSDL file(s) (file(s) with the extension .wsdl). The BPEL parser then creates a set of Handle objects from this input, which represents the BPEL4WS process described by those files.

The input for the EPNML parser is an EPNML file (file with the extension .pnml). The EPNML parser then creates a set of Handle objects in a similar way as is done in the BPEL4WS parser. The result of the parser represents the process(es) described in the EPNML file.

The tool uses the set of handles to create a graphical representation of the BPEL4WS process using the handles of the TM and MM. After that the user can modify the model.

Note: There is one restriction to the use of an EPNML file in the tool, which is that the EPNML file that is parsed must be in conformance with the TM and MM. If the file is not in conformance with the TM and MM, then the file cannot be loaded into the PN-BPEL4WS Tool and the tool gives an overview of what should be altered in the file before it can be used in the tool.

3.2.3. BPEL4WS and EPNML creators

The third step in the creation of the tool consisted of the creation of the BPEL4WS creator and EPNML creator. These creators can be used to respectively generate BPEL4WS files with associated WSDL file(s) and EPNML files of the set of Handle objects that is used in the PN-BPEL4WS Tool.

Each Handle object is described in its BPEL4WS form and in its EPNML form. This means that when the BPEL4WS creator is used, the BPEL4WS creator uses the BPEL4WS form of each Handle object in the set in the creation of the BPEL4WS files with associated WSDL file(s). In the case of the EPNML creator the EPNML form of each Handle object in the set is used in the creation of the EPNML file.

The creator uses the textual representation for each BPEL4WS, WSDL or EPNML object that is used in a Handle object and puts the total result of the translation in a file. In the case that a Handle object contains one or more other Handle objects, then the creator also calls for the textual representation of that handle’s BPEL4WS, WSDL or EPNML objects.

3.2.4. Tool's engine

The fourth step in the creation of the tool consisted of the creation of the GUI environment for the tool. This step is described in the subsection 3.3 of this chapter.

The final step in the creation of the tool consisted of the creation of the engine for the PN-BPEL4WS Tool. The engine connects all previously described parts and adds functionality to use them in the tool.
Figure 3.1 below shows the relations between these parts in the tool, except for the tool's engine, which runs on the background to manage the objects that are created and process the changes made to these objects in the GUI environment of the tool.

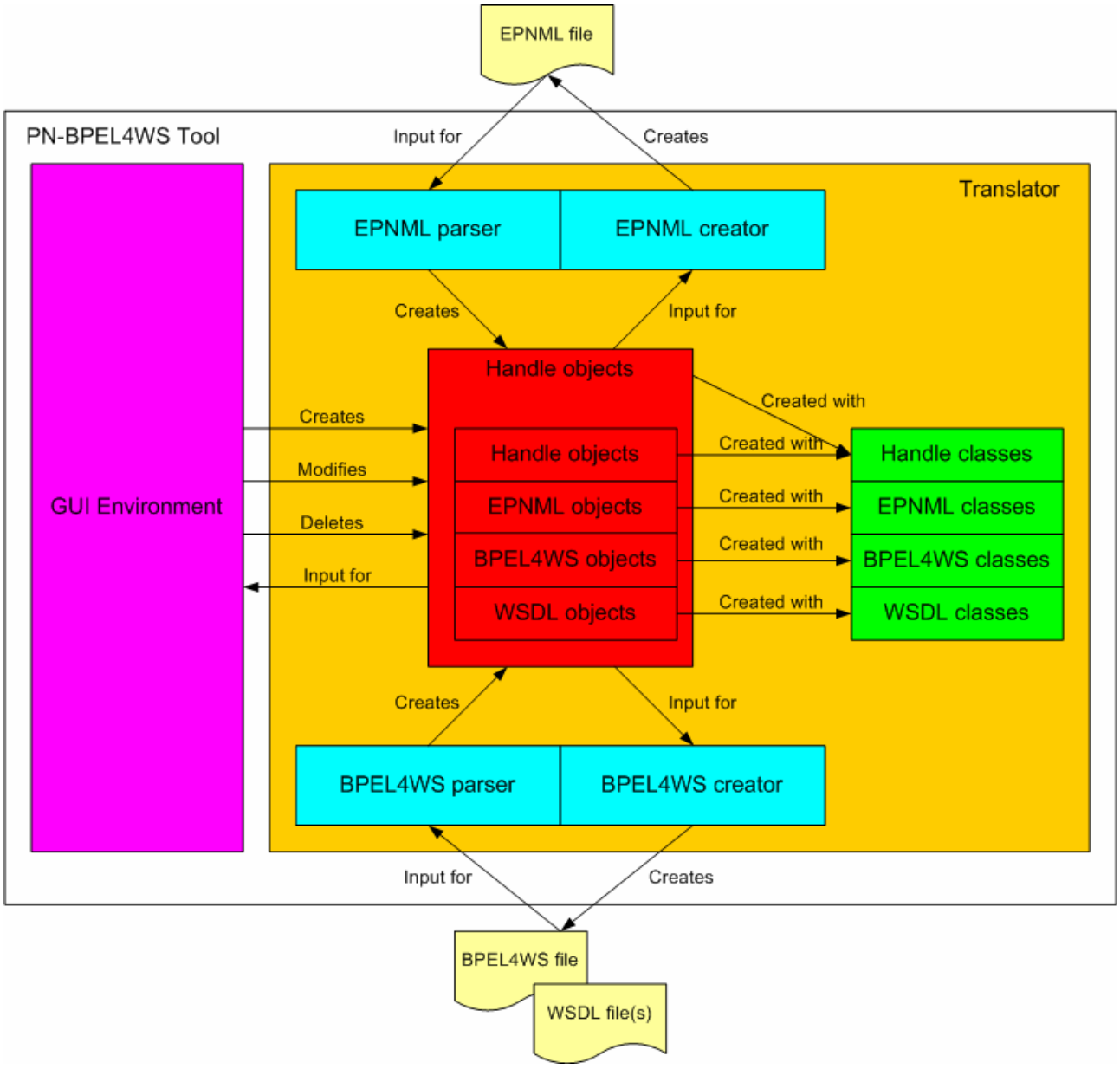


Figure 3.1: Relations between the parts of the PN-BPEL4WS Tool.

3.3. GUI Environment of the PN-BPEL4WS Tool

In this subsection the fourth step in the creation of the tool is described in some more detail than the other parts of the PN-BPEL4WS Tool. The GUI environment of this tool consists of three possible environments each having their own set of features, namely:

- **The "Empty Environment":**
  The "Empty Environment" (shown in figure 3.2 below), is the initial environment one gets when the PN-BPEL4WS Tool is started from the folder it is installed in. In this environment it is not possible to model a (business) process.

*Figure 3.2: The "*Empty Environment*" of the PN-BPEL4WS Tool.*

- **The "Process Environment":**
  The "Process Environment" is the environment one gets when one decides to create / edit a single process. In this environment the user has the possibility to create a new process model from scratch or to modify an already existing process model.
- **The "Project Environment":**
  The "Project Environment" is the environment one gets when one decides to create / edit a BPM. In this environment it is possible to add, modify and delete processes belonging to a single project (business process).

To model a process one either needs to be in the "Process Environment" or in the "Project Environment". The only difference between these two environments is that in the "Project Environment" one has the possibility to add new processes and delete obsolete processes. To change from the "Empty Environment" to the "Process Environment" or the "Project Environment", one needs to create a new process / project or open an already existing process / project.

The “Process Environment” and the “Project Environment” consist of the following main parts (shown in figure 3.3 below, except for the wizards):

- Menu bar.
- Tool bar.
- Working area.
- Status bar.
- Wizards.

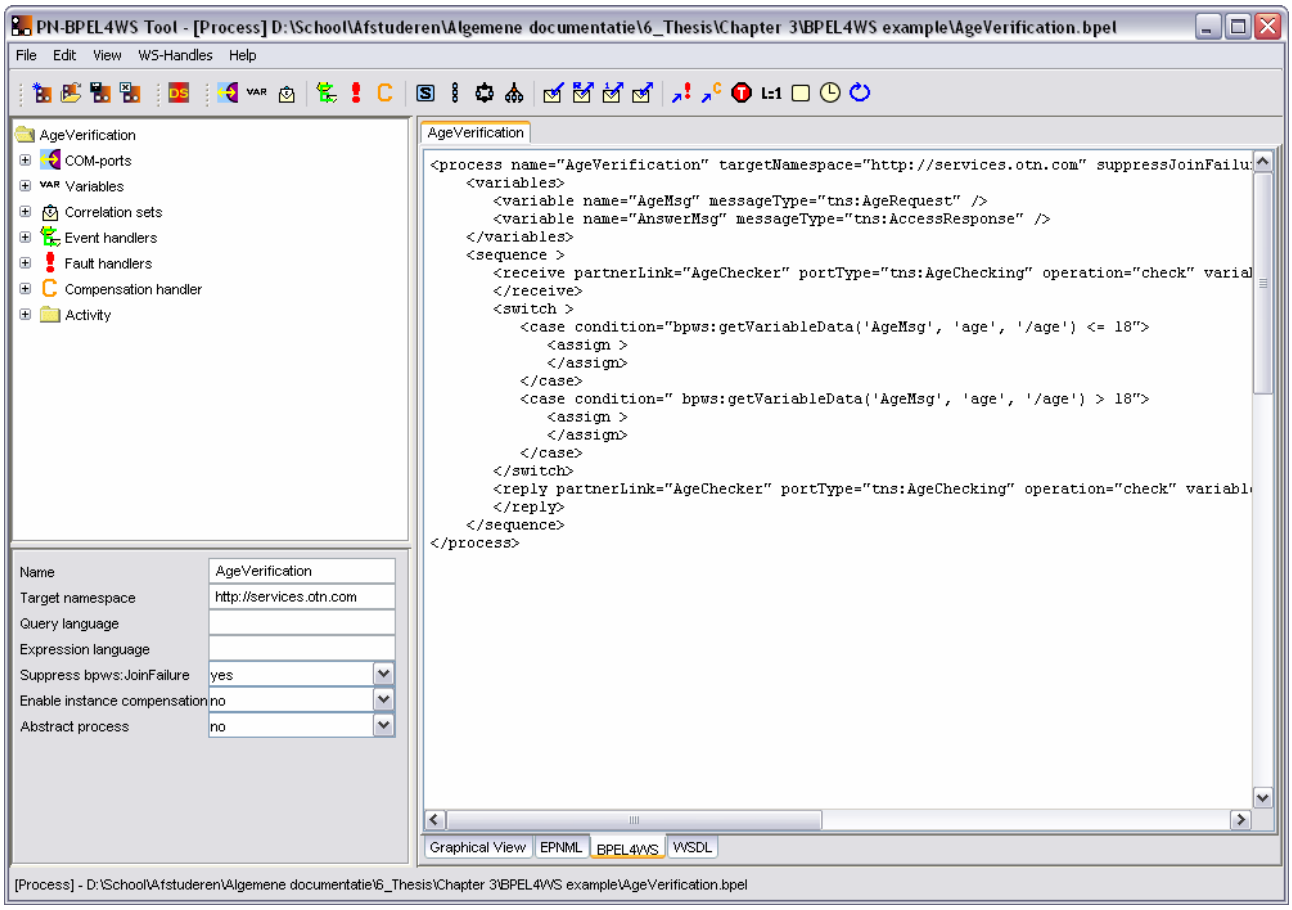


*Figure 3.3: The “*Process Environment*” of the PN-BPEL4WS Tool.*

The menu bar, tool bar and status bar speak for themselves. The menu bar and tool bar contain items which one can use to edit a process in the tool. The most important items in these two bars are the items that are related to the creation of new handles in process models, which are currently active in the tool. The status bar is there to display status messages of the tool’s engine.

The working area is where the modeling of a process takes place. The working area consists of three “inspectors”, namely:

- **Process Inspector** (see figure 3.4):
  This inspector contains all processes that are currently active in the tool. For each process there is a graphical representation, which is editable. Also the translations of the process to EPNML, BPEL4WS and WSDL are displayed for each process, which are not editable.

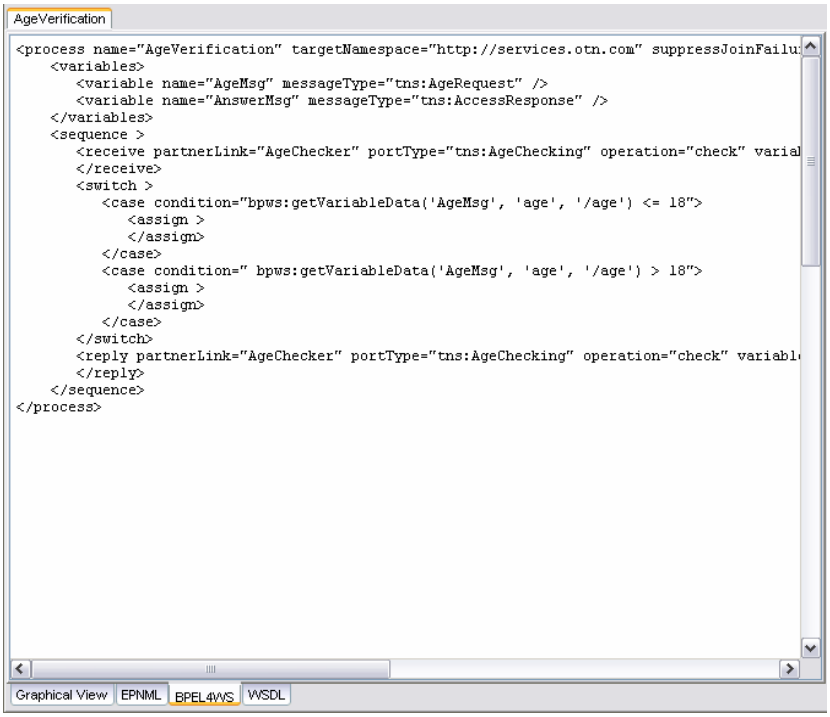


*Figure 3.4: The ´*Process Inspector*" of the PN-BPEL4WS Tool.*

- **Hierarchy Inspector** (see figure 3.5):
  This inspector contains a tree-view of the currently active process(es) in the tool. One can use this tree-view to navigate through the handles of a process and see where they are located in a process.

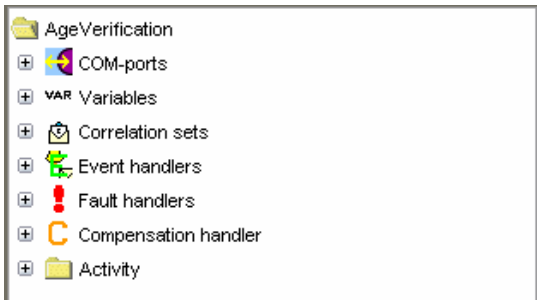


*Figure 3.5: The "*Hierarchy Inspector*" of the PN-BPEL4WS Tool.*

- **Property Inspector** (see figure 3.6):
  This inspector shows the values of the involved parameters for the selected handle in the graphical view of the Process Inspector or for the selected handle in the tree-view of the Hierarchy Inspector.

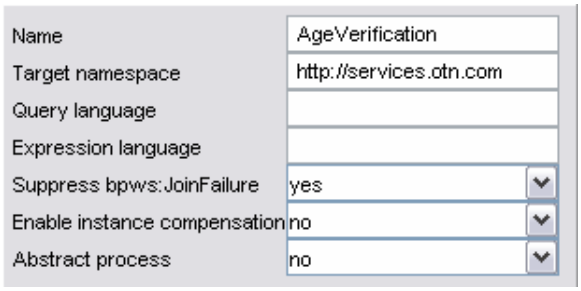


*Figure 3.6: The "*Property Inspector*" of the PN-BPEL4WS Tool.*

Finally there are wizards. The wizards are there to guide the user of the tool. The user can only use these wizards to add new processes to a project and new handles to a process. Each handle has its own wizard. In a wizard the user needs to add values to the involved parameters for each handle as described in the MM.

Note: A quick tutorial how to model with the PN-BPEL4WS Tool is present in the user manual of the tool and therefore is not described in this thesis.

3.3. Relation with the TM and MM

The PN-BPEL4WS Tool is closely related to the TM and MM. The Handle classes as described in subsection 3.2.1 are a translation of the handles that are used in the TM and MM. Also figure 3.1 clearly displays the use of these Handle classes and objects. A Handle object actually is the intermediate state between a single BPEL4WS element and a set of arcs, places and transitions (TM) and between a predefined set of arcs, places and transitions and a single BPEL4WS element (MM).

The TM is also used in the BPEL4WS parser to translate BPEL4WS elements and WSDL elements to Handle objects and in the creation of EPNML files of these Handle objects.

The MM is used by the tool as a guide how to model a Web Services related process.

3.4. Simulation of the PN-BPEL4WS Tool's Result(s)

3.4.1. EPNML and YASPER

This subsection describes the simulation of an EPNML file that has been created with the PN-BPEL4WS Tool. It is not the purpose of this subsection to be a tutorial for the application YASPER. A user guide [YASPER, 2005] is provided with the application, which can serve as tutorial for users that are not acquainted with YASPER.

The application YASPER is a tool in which Petri Net models can be edited and simulated. The following steps have to be taken to open an EPNML file in the tool YASPER that was created by the PN-BPEL4WS Tool:

- Start the application YASPER.
- Open the menu "File" and choose "Open …".
- An "Open file" dialog appears where one can browse to the directory of the EPNML file that has to be opened.
- Click on the desired EPNML file and click the button open of the "Open file" dialog.
- If the contents of the EPNML file are ok, then the tool YASPER will display the contents of that file.

After these steps the EPNML file has been loaded in the tool YASPER. In YASPER one can edit the process model(s) of these EPNML files, but be aware of the fact that one might not be able to open them anymore in the PN-BPEL4WS Tool if the changes to the model are not in conformance with the TM and MM.

To start the simulation of the model one has to add a token to the initial place of the process model. The place with the name "process_<*name of the process*>_start" is the initial place of a process and therefore needs to get an initial token. One can add this token by having your mouse pointer over this place and then perform a right-click with your mouse. A popup window appears from which you select "add a token". The place now holds a black dot inside it.

One can use this simulation to analyze the process model. How one can analyze a model is not a part of this GA and therefore is not described in this thesis. However there are books and papers that describe the analysis for a wide range of models. I refer to the website of the Petri Net Community on the Internet[16] for references to these books and papers.

[16] http://www.informatik.uni-hamburg.de/TGI/PetriNets/

3.4.2. BPEL4WS and Oracle BPEL Process Manager

This subsection describes the simulation of a BPEL4WS file with the associated WSDL file(s) that have been created with the PN-BPEL4WS Tool. It is not the purpose of this section to be a tutorial for the application Oracle BPEL Process Manager[17].

Note: It is only possible to create new BPEL4WS processes in the application Oracle BPEL Designer, which is why I do not describe it in this thesis.

Before the simulation can take place in the Oracle BPEL Process Manager, one has to add some new files in the same directory where the BPEL4WS file is located. These new files are used internally by the Oracle BPEL Process Manager.

One needs to create three files for a single BPEL4WS process:

- **Project file** : This file contains information about the project and has a standard form as described in code 3.1 below. The file always is named ".project".

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
	<name>name of the process</name>
	<comment></comment>
	<projects>
	<projects/>
	<buildSpec>
		<buildCommand>
			<name>bpelz.BPELBuilder</name>
			<arguments></arguments>
		</buildCommand>
	</buildSpec>
	<natures>
		<nature>bpelz.BPELNature</nature>
	</natures>
</projectDescription>
```

Code 3.1: Definition of ".project" file for a BPEL4WS process.

- **BPEL file** : This file contains data about the binding of partner links in the BPEL4WS file to their partner link types in the associated WSDL file(s). The file is always named "bpel.xml". In code 3.2 below the standard form of this file is shown.

```
<?xml version="1.0"?>
<BPELSuitcase>
	<BPELProcess src="name of the BPEL4WS file" id="name of the process">
		<partnerLinkBindings>
		<!-- For each partner link that is used in the BPEL4WS process
			one needs to add a <partnerLinkBinding> element.
		-->
			<partnerLinkBinding name="name of a partner link">
				<property name="wsdlLocation">
					location of the associated WSDL file
				</property>
			</partnerLinkBinding>
		</partnerLinkBindings>
	</BPELProcess>
</BPELSuitcase>
```

Code 3.2: Standard file "bpel.xml" for a BPEL4WS file.

[17] Tutorials for the Oracle BPEL Process Manager as well as the Oracle BPEL Designer and the applications themselves can be found on http://otn.oracle.com/bpel

- **Build file** : This file contains data about the way the BPEL4WS process should be build and has the following standard form as described by code 3.3 below. The file is always named “build.xml”.

```
<?xml version=”1.0”?>
<!-- Run obant on this file to build, package and deploy the name of the
      process BPEL process
-->
<project name=”name of the process” default=”main” basedir=”.”>
      <!-- Name of the domain the generated BPEL suitcase will be deployed
            to
      -->
      <property name=”deploy” value=”default”/>
      <!-- What version number should be used to tag the generated BPEL
            archive?
      -->
      <property name=”rev” value=”1.0”/>
      <target name=”main”>
            <!-- the bpelc task compiles and package BPEL processes into
                  versioned BPEL archives (bpel_...jar). See the
                  “Programming BPEL” guide for more information on the
                  options of this task.
            -->
            <bpelc home=”${home}” rev=”${rev}” deploy=”${deploy}”/>
      </target>
</project>
```

Code 3.3: Standard file “build.xml” for a BPEL4WS file.

Note: If there are more than one BPEL4WS process that one wants to simulate, one needs to change the build file accordingly. How this can be done can be found in the tutorial of the Oracle BPEL Process Manager.

Now one can start the simulation of the BPEL4WS process which has been modeled with the PN-BPEL4WS Tool. The following steps enable the simulation of a BPEL4WS process in the Oracle BPEL Process Manager:

- Start the BPEL Process Manager.
- Open a Command prompt (**Windows taskbar** – **Start** – **Run** – enter “**cmd**” and click the button).
- Go to the directory in which the BPEL4WS file resides and run “obant” in that directory.
- Open a browser and connect to the BPEL Console (http://localhost:9700/BPELConsole).
- Login and select the BPEL4WS process in the “Dashbord” tab of the BPEL Console.
- One now can start simulating the process as described in the Quick Start Tutorial for the Oracle BPEL Process Manager [ORACLE, 2004].

Note: If a BPEL4WS process invokes a new process, the process may halt at that point because the requested process is not active.

Note: One can also analyze the process using the BPEL Console. Information about how to analyze a BPEL4WS process in this console can be found in the documentation belonging to the Oracle BPEL Process Manager application.

Chapter 4: Documentation System

4.1. Introduction

In this chapter the Documentation System (hereafter abbreviated as *DS*) is described in short. The DS is a side result of the GA, which means that originally it did not belong to the planned result of the GA. The DS has been created to support the GA's result. The first four subjects described below cover the main topics of the GA.

The DS consists of a set of documents that are related to the GA. The DS contains information about the following five subjects:

1. **The BPEL4WS & WSDL specifications:**
   This part of the DS contains information about the BPEL4WS specification and the WSDL specification.
2. **The GML of HLPN in this GA and the use of EPNML:**
   This part of the DS describes the graphical notation of HLPN as it is used in this GA. Also one can find information about the EPNML standard that is used in this GA to define the graphical representation of a BPEL4WS process.
3. **The Translation Model and the Mapping Model:**
   In this part one can get information about the Translation Model and the Mapping Model of the GA. The user of the DS can use this part for the following three purposes:
   a. To translate a BPEL4WS process to its graphical HLPN form manually by using the Translation Model.
   b. To develop a BPEL4WS process and its graphical representation manually by using the handles in the Mapping Model.
   c. Reference material.
4. **The PN-BPEL4WS Tool:**
   In this part of the DS one can find information about the PN-BPEL4WS Tool. The following information is provided in this part of the DS: a user manual and a technical manual.
5. **The Documentation System:**
   In this part of the DS one can find information about the DS itself. Information about the following topics can be found in this part:
   a. How to use the DS.
   b. How to maintain the DS.
   c. How to expand the DS.

In this chapter the following topics concerning the DS are described:

- Audience : Describes the kind of audience that is expected to use the DS.
- Purpose : Describes some cases in which the DS can be used.
- Construction : Describes the construction of the DS.
- Maintenance & expansion : Describes how-to maintain & expand the DS.
- A quick tour : Describes in short how to use the DS.

4.2. Audience

The kind of audience that is expected to use the DS is the kind of audience that wants:

- Direct access to topics concerning the research area of the GA
- To create a graphical representation of a BPEL4WS process.
- To translate a BPEL4WS process to its EPNML equivalent.
- Background information when using the PN-BPEL4WS tool.
- Information about the GA.

Another kind of audience could be the kind of audience that needs a “quick access” to information on this GA to start a new research on the basis of this GA.

4.3. Purpose

In the first place the purpose of this DS is to provide the user of the PN-BPEL4WS Tool or someone who is interested in the GA with “quick access” to information about the subjects of this GA. In this case a person may use the DS when:

- One needs information about the TM and MM.
- Something is not clear in the translation of a BPEL4WS element to a set of EPNML elements or in the mapping of a set of predefined EPNML elements to their representation in BPEL4WS.
- One needs background information about a BPEL4WS element, a WSDL element or an EPNML element.
- One needs background information about the extensions made to HLPN as they are used in the GA.
- One needs to access the user manual / technical manual of the PN-BPEL4WS Tool, without starting the tool itself.

On the other hand the purpose of the DS can be elevated beyond this GA. A possible new purpose for this DS, may it be in another form than the initial one, is to form a repository for documents related to the research area of this GA. This means that if there are other researches in the same research area, then the documents related to these researches can be added to this DS to create an information retrieval system concerning topics related to research area, which are:

- BPEL4WS
- (High-level) Petri Nets
- PNML, EPNML and other applications of PNML
- Translations / mappings between BPEL4WS and Petri Nets

4.4. Construction

The DS uses a browser, Microsoft's Internet Explorer[18], to display its contents to the user. It has been created using various techniques (HTML, XML, XSLT, CSS, Macromedia Flash and JavaScript).

The main screen of the DS (see figure 4.1 below) consists of three sections, namely:

- **Title** (top frame):
  This part of the screen holds a standard logo (a Flash animation) for the DS.
- **Toolbar** (bottom-left frame):
  This part of the screen can be used to search for a particular topic in the DS. The following search possibilities are present in this toolbar:
  - **Keyword Search** : In this part of the toolbar one can search for documents in the DS containing a particular keyword. One can also constrain the search to a particular subject of the DS.
  - **Alphabetical Search** : In this part of the toolbar one can search for documents in the DS which are related to the chosen first letter. One can also constrain the search to a particular subject of the DS.
- **Content** (bottom-right frame):
  This part displays the contents of a document of the DS. One can alter the content by following a hyperlink inside the document or by choosing a hyperlink in the part "Search Results" of the toolbar of the DS.

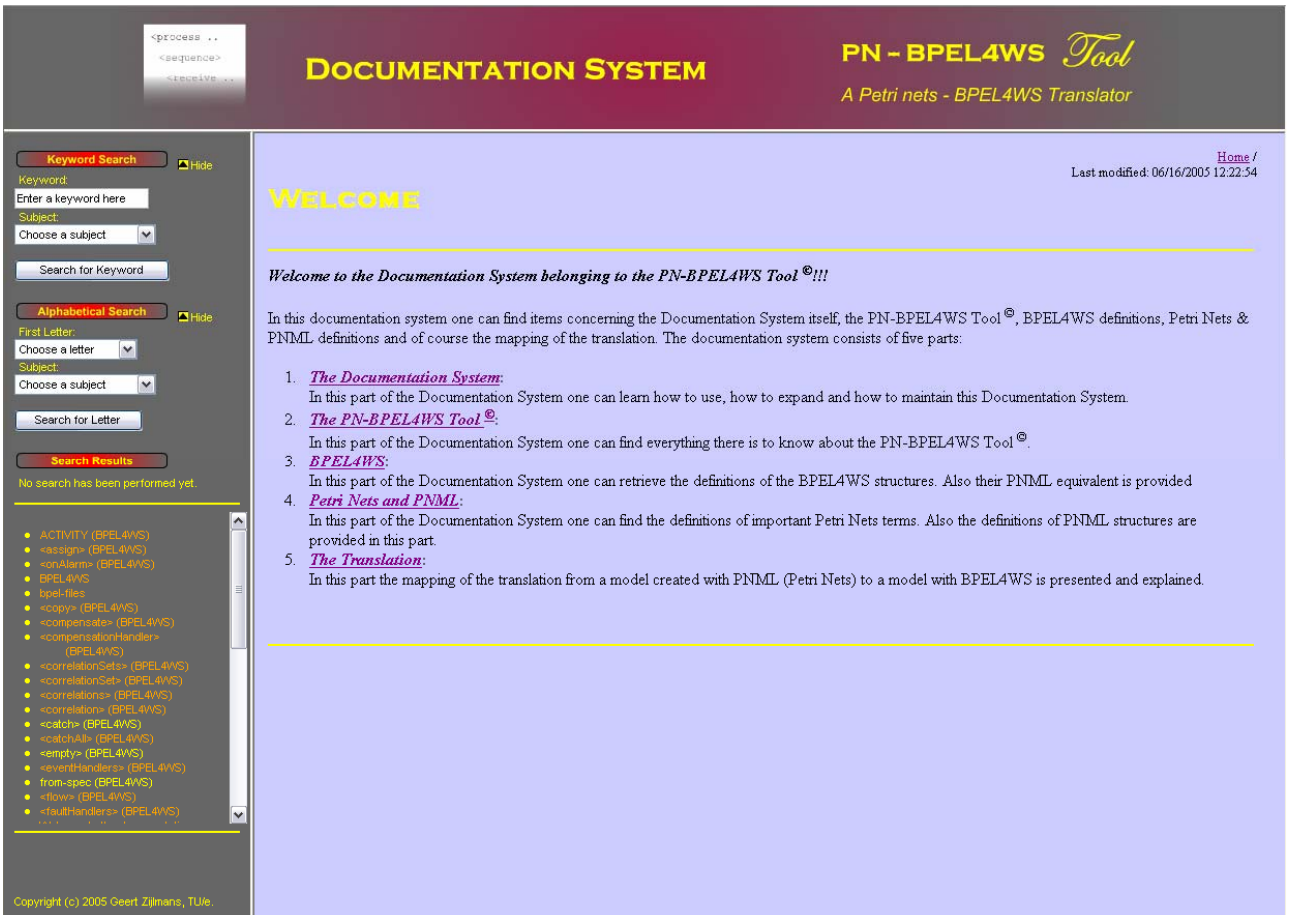


Figure 4.1: "Welcome"-screen of the Documentation System.

[18] Microsoft's Internet Explorer is chosen because of an ActiveX object (Microsoft.XMLDOM) that is used in the toolbar section of the DS.

The hierarchy of the DS' construction can be seen in figure 4.2 below, which clearly shows the three sections described earlier.

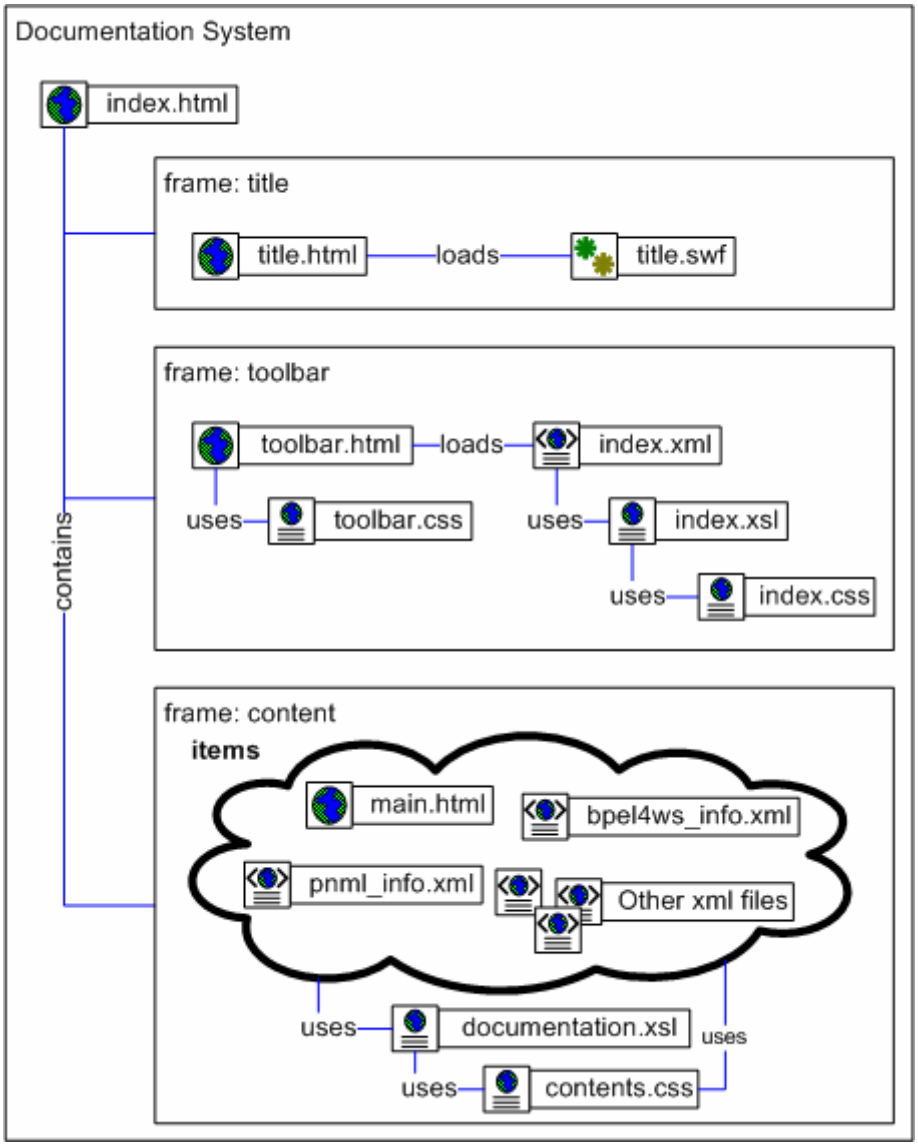


Figure 4.2: The hierarchy of the DS' construction.

4.3.1. Section: Title

In this section the logo of the DS is displayed. The logo itself is a Macromedia Flash animation contained in the file "title.swf". The frame "title" loads the HTML file "title.html", which has an embedded object, referring to the Flash animation of the logo.
Figure 4.3 below shows the graphical representation of the HTML file "title.html" as it is used in the DS.

Figure 4.3: The section "Title" of the DS.

4.3.2. Section: Toolbar

In this section the toolbar of the DS is displayed. The toolbar's main file is the HTML file "toolbar.html". In this file three areas are defined:

- **KEYWORD**:
  This area is related to the "Keyword Search".
- **ALPHABET**:
  This area is related to the "Alphabetical Search".
- **RESULT**:
  This area shows the result of the search.

The "Search"-functionality is enabled by JavaScript. In this file's script the following has been defined:

- Creation of an ActiveX object (Microsoft.XMLDOM) in which the XML file "index.xml" is loaded.
- "Search"-function `loadIndex` that is used to filter the links to the documents of the DS in the XML file "index.xml".
- Functions `hide` and `show` that are respectively used to hide a search area and show a search area (that is hidden).

The XML file "index.xml" holds the following information about a document in the DS:

- Name for the document.
- Path to the location of the document.
- First letter that is related to the document.
- Subject that is related to the document.
- Set of keywords that are related to the document.

One can also view the contents of the XML file "index.xml" outside the DS. The file uses the XSLT file "index.xsl" to transform the contents to their HTML form, which results in a list of links to documents in the DS.
The XSLT file "index.xsl" uses the CSS file "index.css" for applying a style to the result of the transformation.

Figure 4.4 on the right shows the graphical representation of the HTML file "toolbar.html" as it is used in the DS.

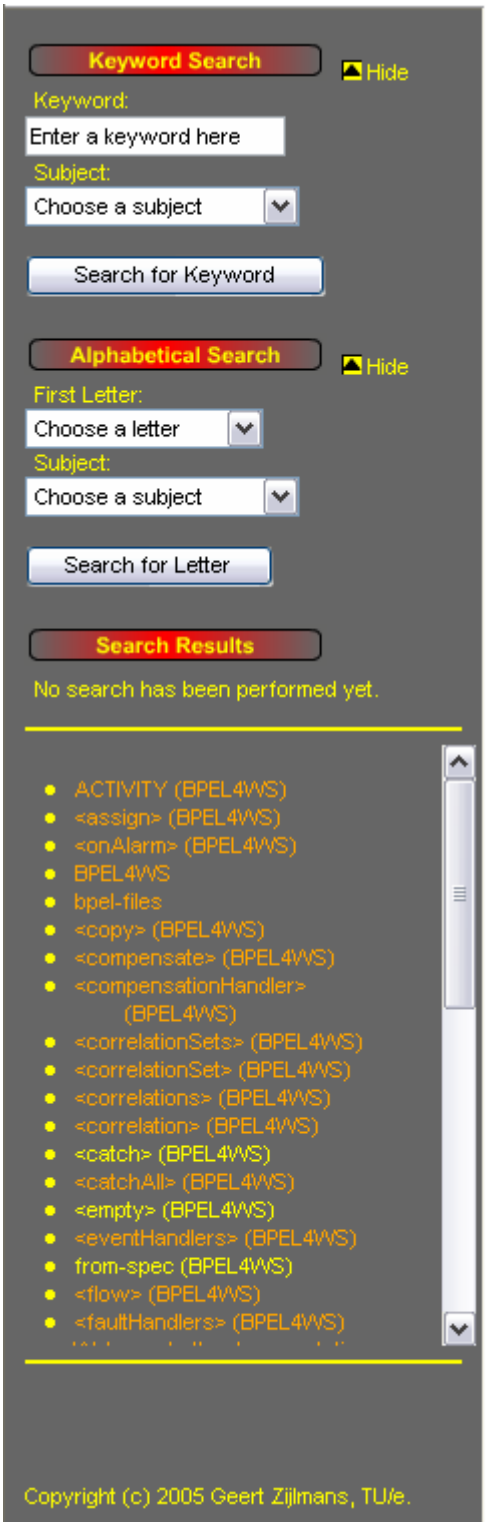


Figure 4.4: The section "Toolbar" of the DS.

4.3.3. Section: Content

In this section the content of a document in the DS is displayed. A document in the DS can either be a HTML file or an XML file. In the first case the file can use the style of the DS, which can be applied to the document by adding a reference to the CSS file “contents.css” in the document. In the latter case the file first needs to be converted to HTML. The conversion is performed using the XSLT file “documentation.xsl”, which also adds a reference to the CSS file “contents.css” to the result of the transformation.

Figure 4.5 below displays the content of the “Welcome”-document, HTML file “main.html”.

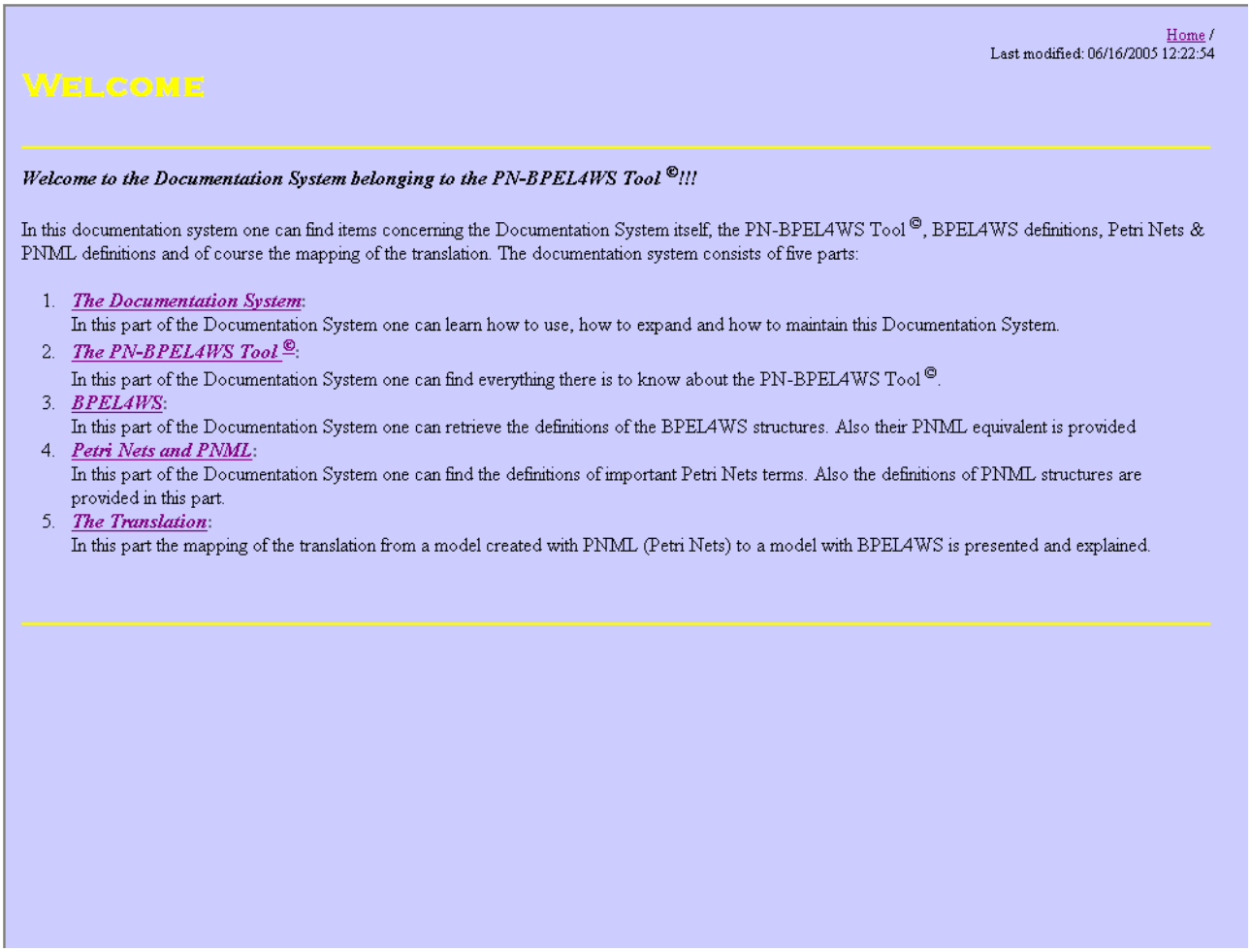

Home /
Last modified: 06/16/2005 12:22:54
WELCOME
Welcome to the Documentation System belonging to the PN-BPEL4WS Tool ©!!!
In this documentation system one can find items concerning the Documentation System itself, the PN-BPEL4WS Tool ©, BPEL4WS definitions, Petri Nets & PNML definitions and of course the mapping of the translation. The documentation system consists of five parts:
1. The Documentation System:
In this part of the Documentation System one can learn how to use, how to expand and how to maintain this Documentation System.
2. The PN-BPEL4WS Tool ©:
In this part of the Documentation System one can find everything there is to know about the PN-BPEL4WS Tool ©.
3. BPEL4WS:
In this part of the Documentation System one can retrieve the definitions of the BPEL4WS structures. Also their PNML equivalent is provided
4. Petri Nets and PNML:
In this part of the Documentation System one can find the definitions of important Petri Nets terms. Also the definitions of PNML structures are provided in this part.
5. The Translation:
In this part the mapping of the translation from a model created with PNML (Petri Nets) to a model with BPEL4WS is presented and explained.

Figure 4.5: The section “Content” of the DS containing the “Welcome”-document.

4.5. Maintenance & Expansion

Maintenance and expansion of a system is always important. This subsection describes the maintenance and expansion operations that are available for the DS.

Maintenance and expansion of the DS consists of the following operations, which are described below:

- Adding new content to the DS.
- Updating content of the DS.
- Deleting content from the DS.

4.4.1. Adding new content to the DS

If one needs to add new content to the DS, one first needs to decide to which of the five subjects it belongs.

The first subject “BPEL4WS & WSDL” and the second subject “The GML of HLPN in this GA and the use of EPNML” mainly contain definitions about the languages themselves, e.g. if one wants to add examples how to use these languages one should add this new content inside these subjects.
New contents to these subjects can be added following the steps described below:

1. Take the template file (XML file “template.xml”) for adding new contents to one of these subjects.
2. Add the information inside the template file.
3. Save it using another name than the one of the template or other present files.
4. Update the files of the subject that have to refer to this new information.
5. Add a reference to the file inside the XML file “index.xml”.

The third subject ”The Translation Model and the Mapping Model” contains the information about the translation of a BPEL4WS process to its graphical HLPN equivalent (in EPNML) and about the modeling of a BPEL4WS process by using handles (graphical representations of BPEL4WS elements). Adding new information to this subject can be done following the steps described below:

1. Open the file belonging to the model:
    a. TM: “translation_model.vsd”;
    b. MM: “mapping_model.vsd”;
2. Insert a new page in the model, where the new information needs to be added.
3. Save the model.
4. Save the model as a “Web Page”.
5. Copy the main file and the folder belonging to the “Web Page” to the folder of the subject in the DS.

The fourth subject “The PN-BPEL4WS Tool” contains a user manual and a technical manual of the tool. If new information needs to be added to one of these manuals, one needs to do the following:

1. Open the file containing the manual:
    a. User manual: “”;
    b. Technical manual: “”;
2. Add the new information to the manual.
3. Save the manual
4. Save the manual as a “Web Page”.
5. Copy the main file and the folder belonging to the “Web Page” to the folder of the subject in the DS.

If new information needs to be added to the last subject “The Documentation System”, one has to follow the same steps as of the first and second subject.

4.4.2. Updating content of the DS

If one needs to update content of the DS, one first needs to decide to which of the five subjects it belongs.

If information needs to be updated of the first, second or fifth subject, one needs to follow the steps described below:

1. Take the file which has to be updated.
2. Update the information inside the file.
3. Save the file.
4. Alter keywords for the reference in the XML file “index.xml” (optional).

If one needs to update information concerning the third or fourth subject, one needs to follow the steps described below:

1. Open the file belonging to the model or manual:
    a. TM: “translation_model.vsd”;
    b. MM: “mapping_model.vsd”;
    c. User manual: “”;
    d. Technical manual: “”;
2. Update the information inside the file.
3. Save the model.
4. Save the model as a “Web Page”.
5. Copy the main file and the folder belonging to the “Web Page” to the folder of the subject in the DS.

4.4.3. Deleting content from the DS

If one needs to delete content from the DS, one first needs to decide to which of the five subjects it belongs.

If information needs to be deleted of the first, second or fifth subject, one needs to follow the steps described below:

1. Take the file which has to be deleted.
2. Delete the file.
3. Update the files in which references are made to the deleted file.
4. Update the XML file “index.xml” by removing the reference to the deleted file.

If one needs to delete information concerning the third or fourth subject, one needs to follow the steps described below:

1. Open the file belonging to the model or manual:
    e. TM: “translation_model.vsd”;
    f. MM: “mapping_model.vsd”;
    g. User manual: “”;
    h. Technical manual: “”;
2. Delete the information inside the file that needs to be deleted.
3. Save the model.
4. Save the model as a “Web Page”.
5. Copy the main file and the folder belonging to the “Web Page” to the folder of the subject in the DS.

4.6. Quick tour

In the “Quick tour” the following topics are described:

- How-to install the DS
- How-to open the DS
- How-to navigate in the DS

4.5.1. How-to install the DS

The DS installs itself automatically when installing the PN-BPEL4WS Tool. If one does not want to install the tool, one can choose to install the DS manually: Extract the file “DS.rar” or “DS.zip” to a folder on your computer.

4.5.2. How-to open the DS

If one is using the PN-BPEL4WS tool, one can follow the next steps:

1. Go to the menu bar.
2. Choose “Help”.
3. Choose “Documentation System” from the list of the menu item “Help”.

A browser will open automatically showing the “Welcome” screen of the DS.

In the other case one can follow the next steps:

1. Go to the folder in which the DS is installed (normally named “DS”).
2. Open the file “DS.html” in a browser of your liking.

The browser will open showing the “Welcome” screen of the DS.

4.5.3. How-to navigate in the DS

In the DS there are two ways to navigate through the contents:

- Using the toolbar of the DS.
- Using links inside the documents.
- Using the “Back” and “Forward” buttons of the browser.

The first way of navigation uses the “Toolbar” section of the screen to navigate through the contents of the DS. The toolbar area “RESULT” contains a list of all documents in the DS that can be related to the search parameters. The section also provides a small search facility on keywords related to the documents and one can choose to view the documents in alphabetical order or on their first letter (see figure 4.4).
If one clicks on a link of a document in the toolbar area “RESULT”, then the content of the chosen document appears inside the section “Content”.

The second way of navigation uses the section “Content”. In some documents links have been added that refer to (information in) another document. By clicking on such a link one can navigate to that particular (part of information in the) document.

The third way of navigation uses the “Back” and “Forward” button of the browser. In the case that one already has viewed several other documents using the other two navigation possibilities, than one can use the “Back” button to reload the document that has been viewed prior to the current one. If one has used the “Back” button, then the “Forward” button will be enabled allowing you to go forward to the document that has been viewed after the current one.

Note: Using the “Back” button too many times may cause the list of results to be displayed in their raw form. A refresh of the page in your browser should fix this problem.

Conclusion

The conclusion of this thesis consists of two parts. In the first part a conclusion is given about the GML of this thesis and in the second part recommendations are made on the basis of the result of this GA.

As one can read in chapter two of this thesis, the approach for creating the GML has been the translation of each BPEL4WS element separately to its form in the GML of HLPN [HLPN, 2000] resulting in the TM and the use of these translations as handles in the MM to develop a WS-related process. The choice for HLPN has been made because of the fact that the semantics and parameters (attributes and sub-elements) of a BPEL4WS element were difficult to translate using only the GML of Classical Petri Nets. The GML of HLPN has more expressive power which allowed me to translate the semantics and parameters of all the BPEL4WS elements. Each BPEL4WS element has been translated on the basis of how they should work, meaning that also the BPEL4WS semantics of the specific element was taken into account during the translation.

The translations themselves are quite simple on the level of their hierarchical transition, but become more complicated when one zooms in on the hierarchical transition of a translation. If one zooms in on the hierarchical transition of a translation one can see the working of the BPEL4WS element, which depends on its semantics and parameters. This fact can be used to verify the working of a BPEL4WS process on element level using the PN-BPEL4WS Tool to translate a BPEL4WS file into an EPNML file, which can be analyzed in a tool that is capable of loading EPNML files. If one has no interest in the exact working of a BPEL4WS element, then one can simply look at the model on its hierarchical level, meaning that one does not go into detail about the actual working of the element knowing that it does what it has to do (e.g. verifying the correct message exchange pattern in which the focus lies on the correct message exchange rather than on the correct execution of a BPEL4WS element in the process).

On the level of their hierarchical transition of the translations my GML is quite similar to the GML as is defined in [WBERG, 2003] and to the GML as is used in Oracle BPEL Designer. Yet these two GMLs do not directly show the connections between all the facets of a WS-related process (communication ports, activities, event handlers, termination, etc.) whereas my GML does show these connections.

Having separate translations for each BPEL4WS element in my GML made it possible to apply compositional modeling, which can be seen as the dragging and dropping of BPEL4WS elements to their correct position in the BPM. This feature simplifies the modeling of a WS-related process drastically and is also used by Oracle BPEL Designer.

Another advantage of having separate translations is the advantage that when changes are made to the BPEL4WS specification [BPEL4WS, 2003] one only has to update a translation which is affected by the changes. In this way my GML can remain consistent even when changes are made to the BPEL4WS specification.

Yet there is a little drawback in my GML, which is that if a process needs to be terminated, one has to check all couplings in the WS-related process model concerning the correct termination of the whole process. The same holds for the fault handling.

The TM, handles, MM and PN-BPEL4WS Tool could be the basis of a standard plug-in for other BPM tools that want to translate a BPEL4WS process to its HLPN form and vice versa. If another tool wants to create a plug-in, then the TM and MM provide enough information concerning the translation and mapping of a WS-related process as described in this thesis and the PN-BPEL4WS Tool's classes of the handles, BPEL4WS elements, WSDL elements and EPNML elements can be reused in the creation of the plug-in.

On the basis of the results of this GA I can make several recommendations concerning EPNML, YASPER and BPEL4WS. First some recommendations concerning EPNML and YASPER, which have already been handed over to Reinier Post, who is one of the authors of the EPNML specification [EPNML, 2004] and a software developer of the tool YASPER at the TU/e:

- The <fill> element of PNML is not supported by EPNML. It would be nice to support this element in EPNML for the coloring of a net as is done in my GML to be able to indicate special places in a net, which could improve the readability of a net.
- The use of reference places in YASPER is quite messy. It is not possible to connect more than one arc to a reference place. The solution in YASPER is to add another reference place inside the hierarchical transition that refers to the same place on the level of the hierarchical transition. Since the figures of this thesis have been modeled with YASPER, one can see the redundancy in reference places (e.g. figure G.1 and G.2, where for each time that the store for the process instances is used in the hierarchical transition "<process> purchaseOrderProcess" (see G.1), one needs to add a new reference place to the store for the process instances (see G.2)).
  Note: In my GML one can connect more than one arc to a single reference place.
- Since it is possible to add tokens to a process modeled in EPNML, it would be nice to give meaning to these tokens. In that case these tokens can be transformed by the transitions that consume them (using the <transformation> element of a transition). All one needs is token type definitions and place typing (= restricting a place to contain only tokens of a certain type). How this can be done, has been described in subsection 2.2.1 "General design decisions" of this thesis.
- In the current EPNML specification a store can only contain a datum. In my opinion a store is a table which can contain records of a certain type. By typing a store, a store can represent a table in which records can be present of the same type. In my GML these records represent a process instance of the process.
- In YASPER it is possible to define a reset-arc, while this type of arc is not supported by the EPNML specification. Include the reset-arc as one of the possible types for an arc as has been done with the inhibitor arc.

Finally some recommendations concerning BPEL4WS:

- If one looks closely to the definitions of the <onAlarm> elements of the <eventHandlers> element and the <pick> element, then one difference between them can be noticed, which is that an <onAlarm> element of an <eventHandlers> element can contain both attributes as can be concluded from its definition (see code C.1) and an <onAlarm> element of an <pick> element can contain only one of the same attributes (see code C.2).

```
<onAlarm for="duration-expression"? until="deadline-expression"?>
      activity
</onAlarm>
```

Code C.1: Definition of a <onAlarm> element for an <eventHandlers> element.

```
<onAlarm (for="duration-expression" | until="deadline-expression")>
      activity
</onAlarm>
```

Code C.2: Definition of a <onAlarm> element for an <pick> element.

  Yet there is a catch, the semantics for an <onAlarm> element of an <eventHandlers> element in state that "Exactly one of these two attributes must occur in any onAlarm event" [BPEL4WS, 2003]. This is a little bit confusing, does it mean that still both attributes may be used or that only one may be used in its definition. In case of the first assumption I would suggest to remove this sentence from the semantics of this element, in the other case I would suggest that the definition becomes the same as that of the <onAlarm> element of a <pick> element.
  Note: My GML uses the second assumption.

- The communication with another process has been defined somewhat awkwardly in my opinion. One has to define a partner link, which involves the definition of a partner link type and the assignments of the roles of the partner link type inside the partner link. In the definition of a partner link type a role is associated with a particular port type that can be used in the communication with the process to which the role belongs. Yet in the definition of a communication activity one has to define a partner link, port type and operation. So what is the goal of defining partner links if they are not really used in the receiving and sending of messages? In my opinion the port type inside the communication activity should be replaced by the role to which this applies (e.g. for a `<receive>` activity element as shown in code C.3 below). In this way the `<partnerLink>` element is used to its full extent.

```
<receive partnerLink=”ncname” role=”(myRole|partnerRole)”
      operation=”” variable=”ncname”? createInstance=”yes|no”?
      standard-attributes>
      standard-elements
      <correlations>
            <correlation set=”” initiate=””/>
      </correlations>
</receive>
```

Code C.3: Altered definition of a `<receive>` activity element.

Note: The use of the value `partnerRole` for the attribute `role` might seem a bit strange in the example above, but one can imagine that one has an asynchronous message exchange between two processes in which prior to the receiving communication activity an invocation of the other process has taken place. In that case one might expect an answer in an output communication port of that process.

The recommendations about BPEL4WS have been sent to the editor, Satish Thatte, of the BPEL4WS specification.

Appendices

APPENDIX A: LIST OF TERMS 97

APPENDIX B: LIST OF ABBREVIATIONS 97

APPENDIX C: BPEL4WS IN BACKUS-NAUR FORM 97

APPENDIX D: BPEL4WS DIVIDED IN CLASSES 97

APPENDIX E: THE TRANSLATION MODEL 97

APPENDIX F: THE MAPPING MODEL 97

APPENDIX G: THE INITIAL EXAMPLE OF [BPEL4WS, 2003] 97

G.1. WSDL FILE 97
G.2. BPEL4WS FILE 97
G.3. TRANSLATION OF THE EXAMPLE USING THE TM 97

Appendix A: List of Terms

Atomic activity
Seen through the eyes of the developer of the BPM this is an ***activity*** in a ***process*** that is an indivisible unit of work. In the *modeling language* Petri Nets an atomic activity can be represented by a transition.

COM-port
A COM-port is a special place in a HLPN model using the GML of this research. It represents a communication link with another process. The kind of token that can be associated with a COM-port must be of the type “message”.

Compositional modeling
In compositional modeling one uses predefined structures to define a BPM. The predefined structures are like the pieces in a jig-saw puzzle. If one knows where a piece belongs in the puzzle, then one can put it in the correct place, connecting it to other pieces of the puzzle. The same counts for the predefined structures.

Graphical modeling language
A graphical modeling language (GML) is a language that uses the standard graphical notation of a modeling language. A GML of a modeling language allows the developer to speed up the development of a BPM in the modeling language.

Halting
The term halting is used in the development and execution of a process to indicate that a process instance does not end entirely. Only the main activity of the process ends, while the event handlers, fault handlers and / or compensation handler of the process may continue as normal.

Modeling language
A modeling language is a certain set of definitions that can be used to express / define a process. Most of the modeling languages have a mathematical background (e.g. Petri Nets).

Start activities
These are activities in a BPEL4WS process that initiate a new instance of the process. If there are two or more start activities concurrently enabled, then these start activities all are associated to a single new process instance (= only one new process instance is created).

Termination
Termination is the ending of the existence of a process instance. A process instance can end in two possible states:

- Normal : In this state the process instance ended as it should have ended.
- Abnormal : In this state something went wrong during the execution of the process instance, which caused the process instance to end abnormally (e.g. fault occurred for the process as a whole or has been ended explicitly).

WS-related process
A WS-related process is a process that uses the Web Services Standards (see figure I.1) to exchange information with other processes.

Appendix B: List of Abbreviations

BNF	Backus-Naur Form
BPEL4WS	Business Process Execution Language for Web Services
BPMN	Business Process Modeling Notation
CSS	Cascading Style Sheets
DS	Documentation System
GA	Graduation Assignment
GML	Graphical Modeling Language
GUI	Graphical User Interface
HLPN	High-level Petri Nets
HTML	Hypertext Markup Language
MM	Mapping Model
PNML	Petri Net Markup Language
TM	Translation Model
TU/e	University of Technology Eindhoven (Department of Mathematics & Computer Science)
WS	Web Services
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
XML	eXtended Markup Language
XSD	XML Schema Datatype(s)
XSLT	eXtensible Stylesheet Language Transformations

Appendix C: BPEL4WS in Backus-Naur Form

The following specification of BPEL4WS in BNF has been developed by Geert Zijlmans.

```
bool-expr          : Boolean expression [XPATH, 1999]
general-expr       : Expression that returns a string, number or Boolean
                     value [XPATH, 1999]
duration-expr      : Expression results in XML Schema type duration values
                     [XPATH, 1999]
deadline-expr      : Expression results in XML Schema type dateTime or date
                     values [XPATH, 1999]
uri                : Uniform Resource Identifier of the process
anyURI             : Uniform Resource Identifier of an external file
ncname             : namespace name [XMLN, 1999]
qname              : qualified name [XMLN, 1999]
qname-list         : List of qname's
```

```
process              ::=   <process top-attributes >
                             partnerLinks?
                             partners?
                             variables?
                             correlationSets?
                             eventHandlers?
                             compensationHandler?
                             faultHandlers?
                             activity
                           </process>

top-attributes       ::=   name="ncname" targetNamespace="uri"
                           queryLanguage="anyURI"?
                           expressionLanguage="anyURI"?
                           suppressJoinFailure="yes|no"?
                           enableInstanceCompensation="yes|no"?
                           abstractProcess="yes|no"?
                           xmlns="http://schemas.xmlsoap.org/
                                       ws/2003/03/business-process/"

partnerLinks         ::=   <partnerLinks>
                             partnerLink+
                           </partnerLinks>

partnerLink          ::=   <partnerLink name="ncname"
                             partnerLinkType="qname"
                             myRole="ncname"?
                             partnerRole="ncname"? />

partners             ::=   <partners>
                             partner+
                           </partners>

partner              ::=   <partner name="ncname">
                             partner-partnerLink+
                           </partner>
```

```
partner-partnerLink     ::=   <partnerLink name="ncname"/>

variables               ::=   <variables>
                                    variable+
                              </variables>

variable                ::=   <variable name="ncname" var-type? />

var-type                ::=   messageType="qname" | type="qname" |
                              element="qname"

correlationSets         ::=   <correlationSets>
                                    correlationSet+
                              </correlationSets>

correlationSet          ::=   <correlationSet name="ncname"
                                    properties="qname-list"/>

eventHandlers           ::=   <eventHandlers>
                                    (onMessage | onAlarm |
                                     onMessage+ onAlarm*)
                              </eventHandlers>

onMessage               ::=   <onMessage partnerLink="ncname"
                                     portType="qname" operation="ncname"
                                     variable="ncname"?>
                                    correlations?
                                    activity
                              </onMessage>

onAlarm                 ::=   <onAlarm (for="duration-expr" |
                                     until="deadline-expr")>
                                    activity
                              </onAlarm>

compensationHandler     ::=   <compensationHandler>
                                    activity
                              </compensationHandler>

faultHandlers           ::=   <faultHandlers>
                                    catch*
                                    catchAll?
                              </faultHandlers>

catch                   ::=   <catch faultName="qname"?
                                     faultVariable="ncname"?>
                                    activity
                              </catch>

catchAll                ::=   <catchAll>
                                    activity
                              </catchAll>

activity                ::=   basic activity | structured activity

basic activity          ::=   receive | reply | invoke | assign | empty |
                              terminate | throw | compensate | wait
```

```
structured activity     ::=   scope | flow | sequence | while | pick |
                              switch

standard-attributes     ::=   name="ncname"? joinCondition="bool-expr"?
                              suppressJoinFailure="yes|no"?

standard-elements       ::=   <target linkName="ncname"/>*
                              <source linkName="ncname"
                               transitionCondition="bool-expr"? />*

correlations            ::=   <correlations>
                                    correlation+
                              </correlations>

correlation             ::=   <correlation set="ncname" initiate="yes|no"?
                                    pattern="in|out|out-in"?/>

receive                 ::=   <receive partnerLink="ncname"
                                     portType="qname" operation="ncname"
                                     variable="ncname"?
                                     createInstance="yes|no"?
                                     standard-attributes>
                                    standard-elements
                                    correlations?
                              </receive>

reply                   ::=   <reply partnerLink="ncname"
                                     portType="qname" operation="ncname"
                                     variable="ncname"? faultName="qname"?
                                     standard-attributes>
                                    standard-elements
                                    correlations?
                              </reply>

invoke                  ::=   <invoke partnerLink="ncname"
                                     portType="qname" operation="ncname"
                                     inputVariable="ncname"?
                                     outputVariable="ncname"?
                                     standard-attributes>
                                    standard-elements
                                    correlations?
                                    catch*
                                    catchAll?
                                    compensationHandler?
                              </invoke>

assign                  ::=   <assign standard-attributes>
                                    standard-elements
                                    copy+
                              </assign>

copy                    ::=   <copy>
                                    from-spec
                                    to-spec
                              </copy>
```

from-spec ::= <from variable="ncname" part="ncname"?/> |
<from partnerLink="ncname"
endpointReference="myRole|partnerRole"/> |
<from variable="ncname" property="qname"/> |
<from expression="general-expr"/> |
<from> ...literal value ... </from> |
<from variable="ncname" part="ncname"?
query="queryString"?/> *(only in executable process)* |
<from opaque="yes"/> *(only in abstract process)*

to-spec ::= <to variable="ncname" part="ncname"?/> |
<to partnerlink="ncname"/> |
<to variable="ncname" property="qname"/> |
<to variable="ncname" part="ncname"?
query="queryString"?/> (*only in executable process)*

empty ::= <empty **standard-attributes**>
standard-elements
</empty>

terminate ::= <terminate **standard-attributes**>
standard-elements
</terminate> (*only in executable process)*

throw ::= <throw faultName="qname"
faultVariable="ncname"? **standard-attributes**>
standard-elements
</throw>

compensate ::= <compensate scope="ncname"? **standard-attributes**>
standard-elements
</compensate>

wait ::= <wait (for="duration-expr" |
until="deadline- expr") **standard-attributes**>
standard-elements
</wait>

scope ::= <scope variableAccessSerializable="yes|no"
standard-attributes>
standard-elements
variables?
correlationSets?
faultHandlers?
compensationHandler?
eventHandlers?
activity
</scope>

```
flow        ::=  <flow standard-attributes>
                     standard-elements
                     links?
                     activity+
                 </flow>

links       ::=  <links>
                     link+
                 </links>

link        ::=  <link name="ncname"/>

sequence    ::=  <sequence standard-attributes>
                     standard-elements
                     activity+
                 </sequence>

while       ::=  <while condition="bool-expr" standard-
                  attributes>
                     standard-elements
                     activity
                 </while>

pick        ::=  <pick createInstance="yes|no"? standard-
                  attributes>
                     onMessage+
                     onAlarm*
                 </pick>

switch      ::=  <switch standard-attributes>
                     standard-elements
                     case+
                     otherwise?
                 </switch>

case        ::=  <case condition="bool-expr">
                     activity
                 </case>

otherwise   ::=  <otherwise>
                     activity
                 </otherwise>
```

Listing C.1: BPEL4WS [BPEL4WS, 2003] elements and attributes in BNF.

Appendix D: BPEL4WS divided in Classes

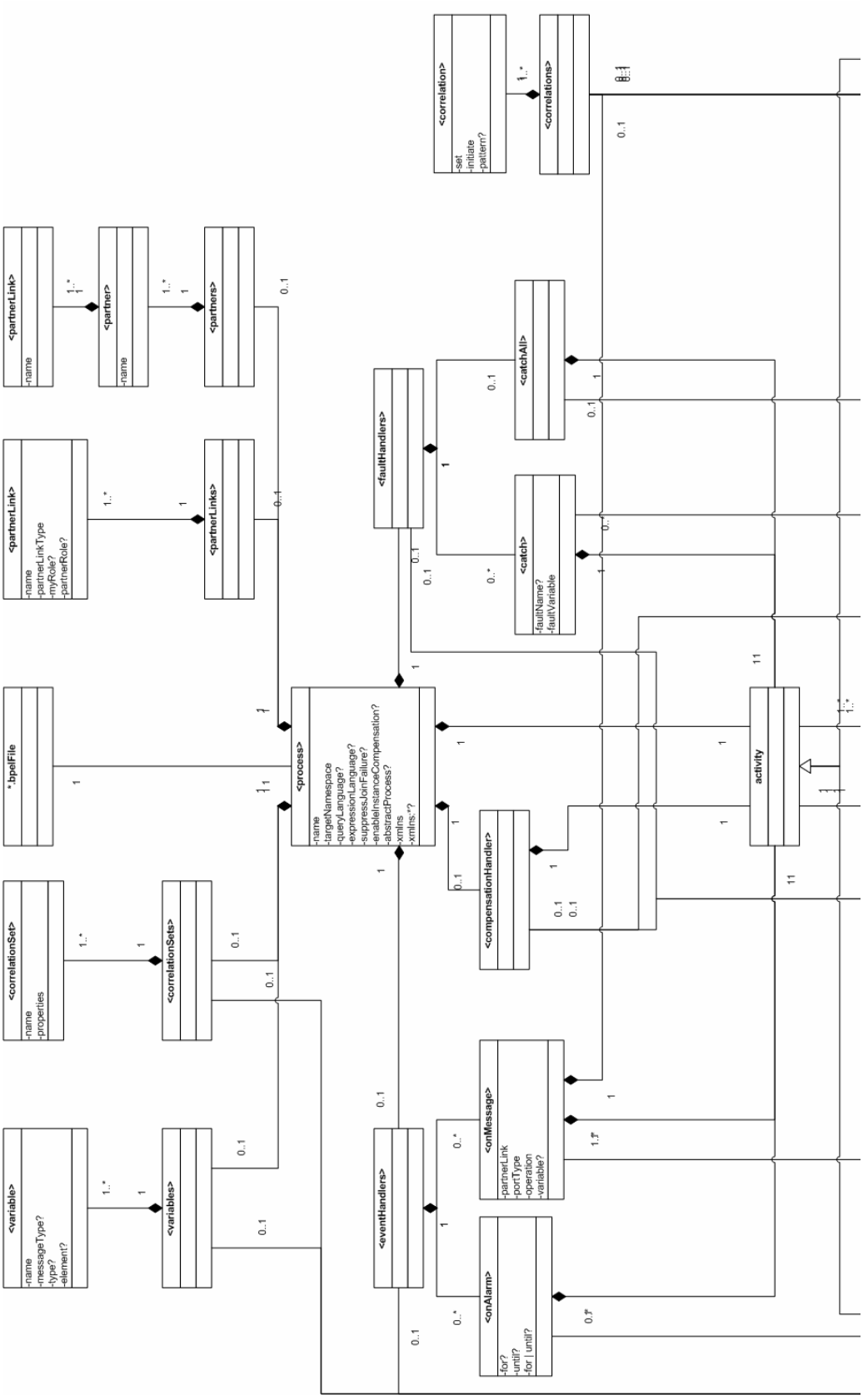


Figure D.1: The BPEL4WS class diagram.

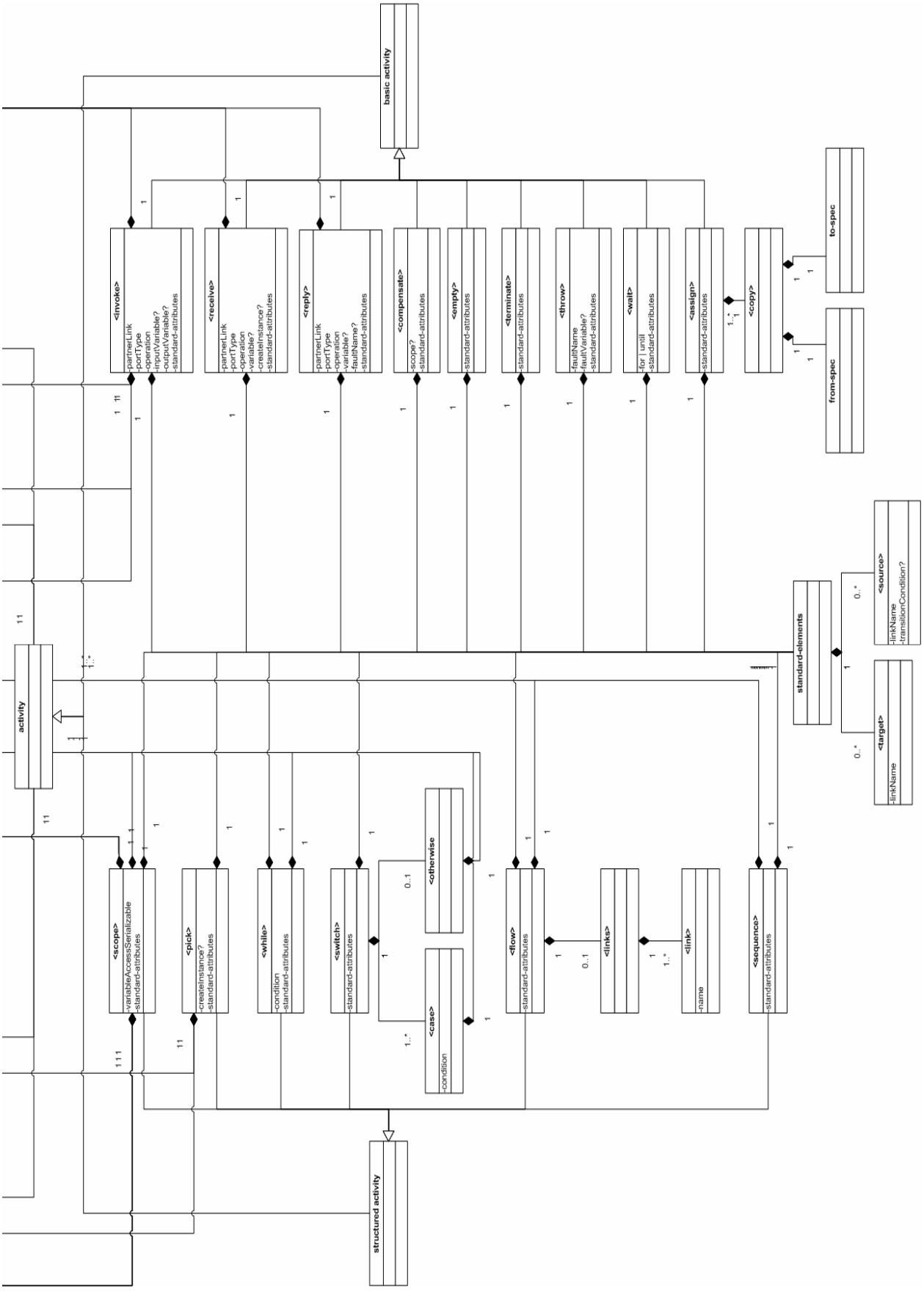


Figure D.1 (continued): The BPEL4WS class diagram.

Appendix E: The Translation Model

Appendix F: The Mapping Model

Appendix G: The Initial Example of [BPEL4WS, 2003]

G.1. WSDL file

```
<definitions targetNamespace="http://manufacturing.org/wsdl/purchase"
             xmlns:sns="http://manufacturing.org/xsd/purchase"
             xmlns:pos="http://manufacturing.org/wsdl/purchase"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">

       <import namespace="http://manufacturing.org/xsd/purchase"
              location="http://manufacturing.org/xsd/purchase.xsd"/>


       <message name="POMessage">
              <part name="customerInfo" type="sns:customerInfo"/>
              <part name="purchaseOrder" type="sns:purchaseOrder"/>
       </message>

       <message name="InvMessage">
              <part name="IVC" type="sns:Invoice"/>
       </message>

       <message name="orderFaultType">
              <part name="problemInfo" type="xsd:string"/>
       </message>

       <message name="shippingRequestMessage">
              <part name="customerInfo" type="sns:customerInfo"/>
       </message>

       <message name="shippingInfoMessage">
              <part name="shippingInfo" type="sns:shippingInfo"/>
       </message>

       <message name="scheduleMessage">
              <part name="schedule" type="sns:scheduleInfo"/>
       </message>


       <!-- portTypes supported by the purchase order process -->
       <portType name="purchaseOrderPT">
              <operation name="sendPurchaseOrder">
                     <input message="pos:POMessage"/>
                     <output message="pos:InvMessage"/>
                     <fault name="cannotCompleteOrder"
                            message="pos:orderFaultType"/>
              </operation>
       </portType>

       <portType name="invoiceCallbackPT">
              <operation name="sendInvoice">
                     <input message="pos:InvMessage"/>
              </operation>
       </portType>

       <portType name="shippingCallbackPT">
              <operation name="sendSchedule">
                     <input message="pos:scheduleMessage"/>
              </operation>
       </portType>
```

```
	<!-- portType supported by the invoice services -->
	<portType name="computePricePT">
		<operation name="initiatePriceCalculation">
			<input message="pos:POMessage"/>
		</operation>
		<operation name="sendShippingPrice">
			<input message="pos:shippingInfoMessage"/>
		</operation>
	</portType>

	<!-- portType supported by the shipping service -->
	<portType name="shippingPT">
		<operation name="requestShipping">
			<input message="pos:shippingRequestMessage"/>
			<output message="pos:shippingInfoMessage"/>
			<fault name="cannotCompleteOrder"
				message="pos:orderFaultType"/>
		</operation>
	</portType>

	<!-- portType supported by the production scheduling process -->
	<portType name="schedulingPT">
		<operation name="requestProductionScheduling">
			<input message="pos:POMessage"/>
		</operation>
		<operation name="sendShipingSchedule">
			<input message="pos:scheduleMessage"/>
		</operation>
	</portType>


	<plnk:partnerLinkType name="purchasingLT">
		<plnk:role name="purchaseService">
			<plnk:portType name="pos:purchaseOrderPT"/>
		</plnk:role>
	</plnk:partnerLinkType>

	<plnk:partnerLinkType name="invoicingLT">
		<plnk:role name="invoiceService">
			<plnk:portType name="pos:computePricePT"/>
		</plnk:role>
		<plnk:role name="invoiceRequester">
			<plnk:portType name="pos:invoiceCallbackPT"/>
		</plnk:role>
	</plnk:partnerLinkType>

	<plnk:partnerLinkType name="shippingLT">
		<plnk:role name="shippingService">
			<plnk:portType name="pos:shippingPT"/>
		</plnk:role>
		<plnk:role name="shippingRequester">
			<plnk:portType name="pos:shippingCallbackPT"/>
		</plnk:role>
	</plnk:partnerLinkType>

	<plnk:partnerLinkType name="schedulingLT">
		<plnk:role name="schedulingService">
			<plnk:portType name="pos:schedulingPT"/>
		</plnk:role>
	</plnk:partnerLinkType>
</definitions>
```

Code G.1: WSDL file of the initial example of [BPEL4WS, 2003].

G.2. BPEL4WS file

```
<process name="purchaseOrderProcess"
             targetNamespace="http://acme.com/ws-bp/purchase"
             xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
             xmlns:lns="http://manufacturing.org/wsdl/purchase">

       <partnerLinks>
             <partnerLink name="purchasing" partnerLinkType="lns:purchasingLT"
                    myRole="purchaseService"/>
             <partnerLink name="invoicing" partnerLinkType="lns:invoicingLT"
                    myRole="invoiceRequester" partnerRole="invoiceService"/>
             <partnerLink name="shipping" partnerLinkType="lns:shippingLT"
                    myRole="shippingRequester" partnerRole="shippingService"/>
             <partnerLink name="scheduling" partnerLinkType="lns:schedulingLT"
                    partnerRole="schedulingService"/>
       </partnerLinks>


       <variables>
             <variable name="PO" messageType="lns:POMessage"/>
             <variable name="Invoice" messageType="lns:InvMessage"/>
             <variable name="POFault" messageType="lns:orderFaultType"/>
             <variable name="shippingRequest"
                    messageType="lns:shippingRequestMessage"/>
             <variable name="shippingInfo" messageType="lns:shippingInfoMessage"/>
             <variable name="shippingSchedule" messageType="lns:scheduleMessage"/>
       </variables>


       <faultHandlers>
             <catch faultName="lns:cannotCompleteOrder" faultVariable="POFault">
                    <reply partnerLink="purchasing" portType="lns:purchaseOrderPT"
                           operation="sendPurchaseOrder" variable="POFault"
                           faultName="cannotCompleteOrder"/>
             </catch>
       </faultHandlers>


       <sequence>

             <receive partnerLink="purchasing" portType="lns:purchaseOrderPT"
                    operation="sendPurchaseOrder" variable="PO">
             </receive>

             <flow>

                    <links>
                           <link name="ship-to-invoice"/>
                           <link name="ship-to-scheduling"/>
                    </links>

                    <sequence>

                           <assign>
                                  <copy>
                                         <from variable="PO" part="customerInfo"/>
                                         <to variable="shippingRequest"
                                                part="customerInfo"/>
                                  </copy>
                           </assign>
```

```
                        <invoke partnerLink="shipping"
                               portType="lns:shippingPT"
                               operation="requestShipping"
                               inputVariable="shippingRequest"
                               outputVariable="shippingInfo">
                               <source linkName="ship-to-invoice"/>
                        </invoke>

                        <receive partnerLink="shipping"
                               portType="lns:shippingCallbackPT"
                               operation="sendSchedule"
                               variable="shippingSchedule">
                               <source linkName="ship-to-scheduling"/>
                        </receive>

                </sequence>

                <sequence>

                        <invoke partnerLink="invoicing"
                               portType="lns:computePricePT"
                               operation="initiatePriceCalculation"
                               inputVariable="PO">
                        </invoke>

                        <invoke partnerLink="invoicing"
                               portType="lns:computePricePT"
                               operation="sendShippingPrice"
                               inputVariable="shippingInfo">
                               <target linkName="ship-to-invoice"/>
                        </invoke>

                        <receive partnerLink="invoicing"
                               portType="lns:invoiceCallbackPT"
                               operation="sendInvoice" variable="Invoice"/>

                </sequence>

                <sequence>

                        <invoke partnerLink="scheduling"
                               portType="lns:schedulingPT"
                               operation="requestProductionScheduling"
                               inputVariable="PO">
                        </invoke>

                        <invoke partnerLink="scheduling"
                               portType="lns:schedulingPT"
                               operation="sendShippingSchedule"
                               inputVariable="shippingSchedule">
                               <target linkName="ship-to-scheduling"/>
                        </invoke>

                </sequence>

        </flow>

        <reply partnerLink="purchasing" portType="lns:purchaseOrderPT"
                operation="sendPurchaseOrder" variable="Invoice"/>

    </sequence>

</process>
```

Code G.2: BPEL4WS file of the initial example of [BPEL4WS, 2003].

G.3. Translation of the example using the TM

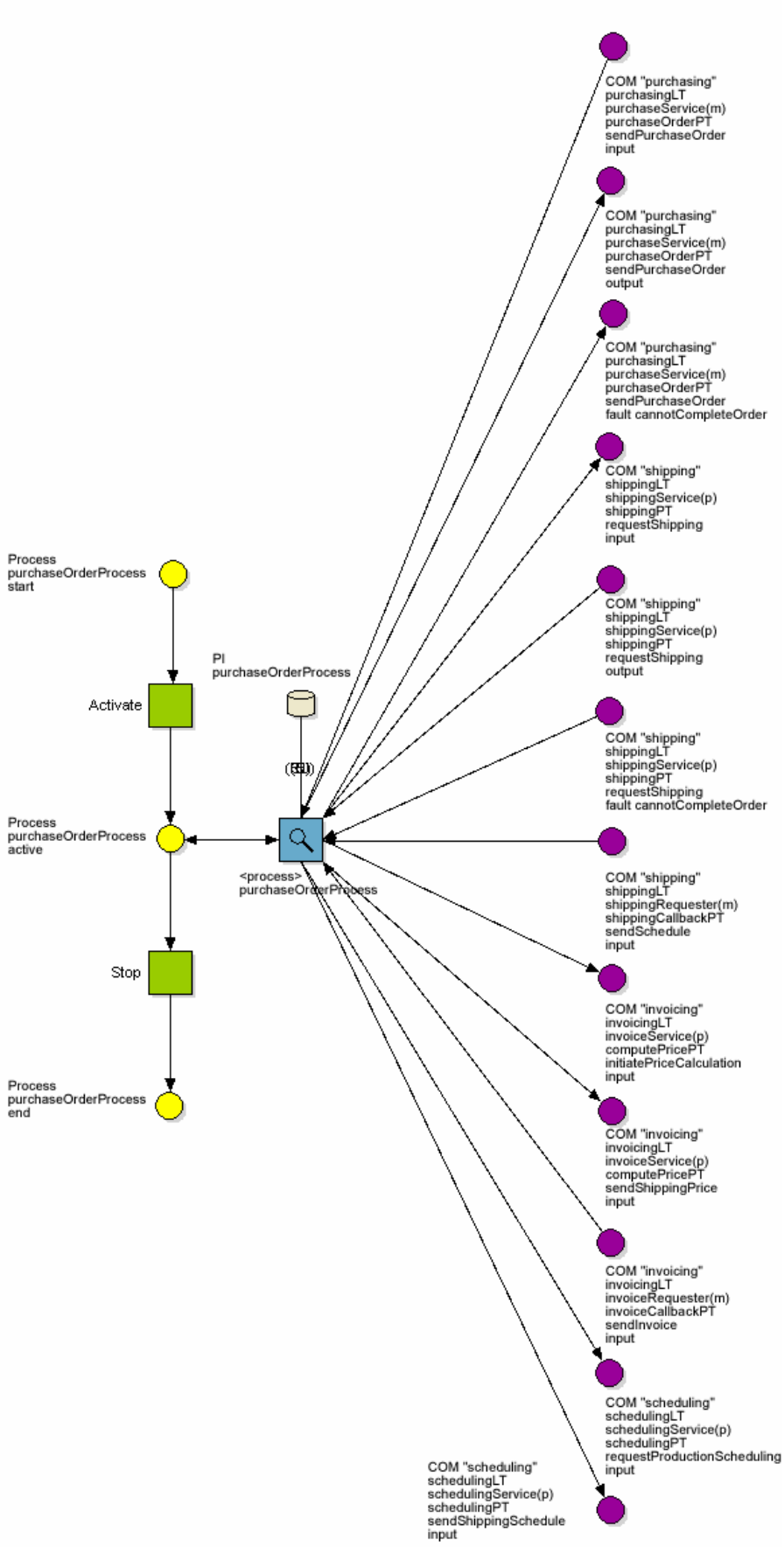


Figure G.1: Graphical representation of the process "purchaseOrderProcess".

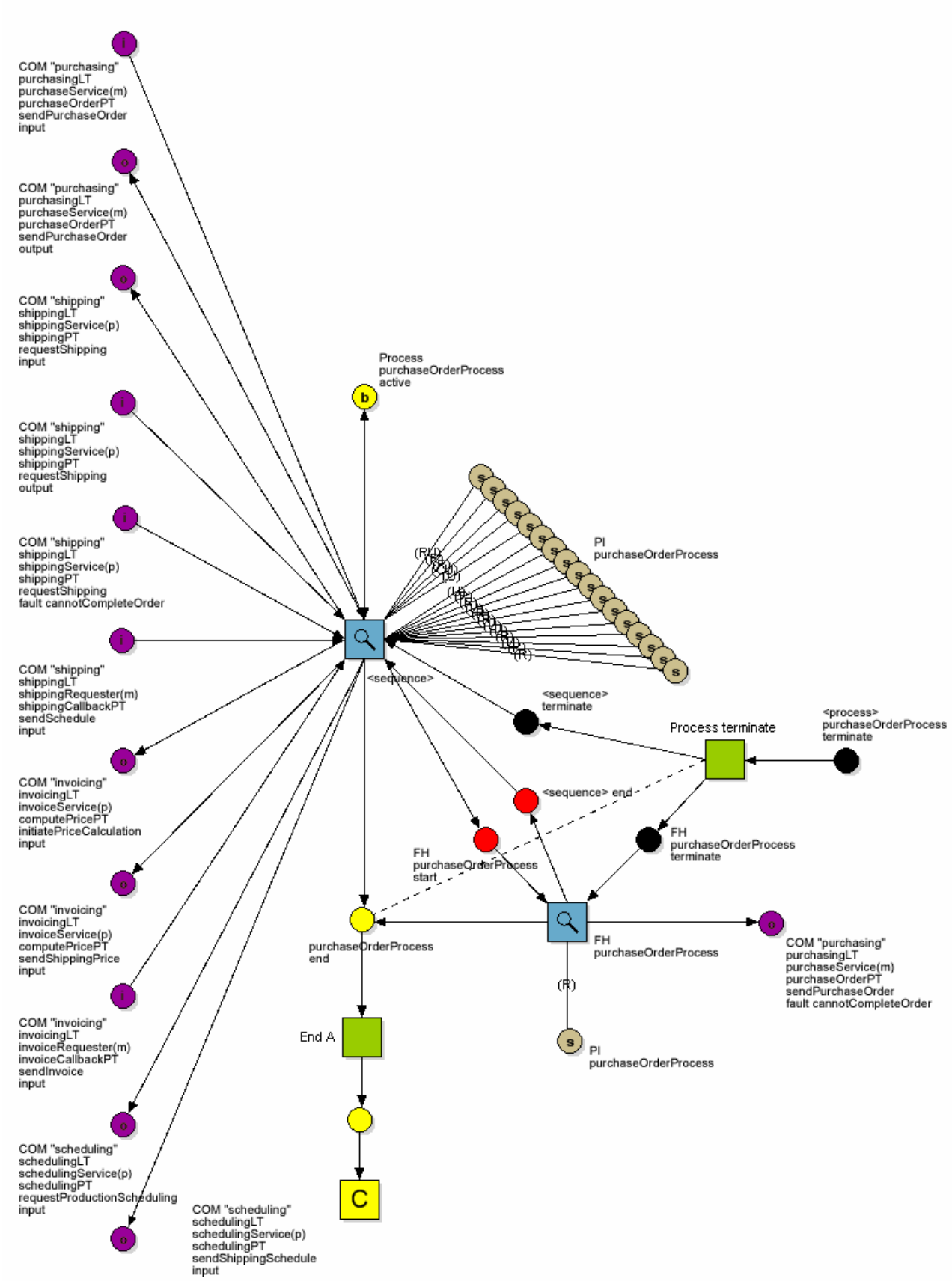


Figure G.2: Zooming in on the "<process> purchaseOrderProcess" element of figure G.1.

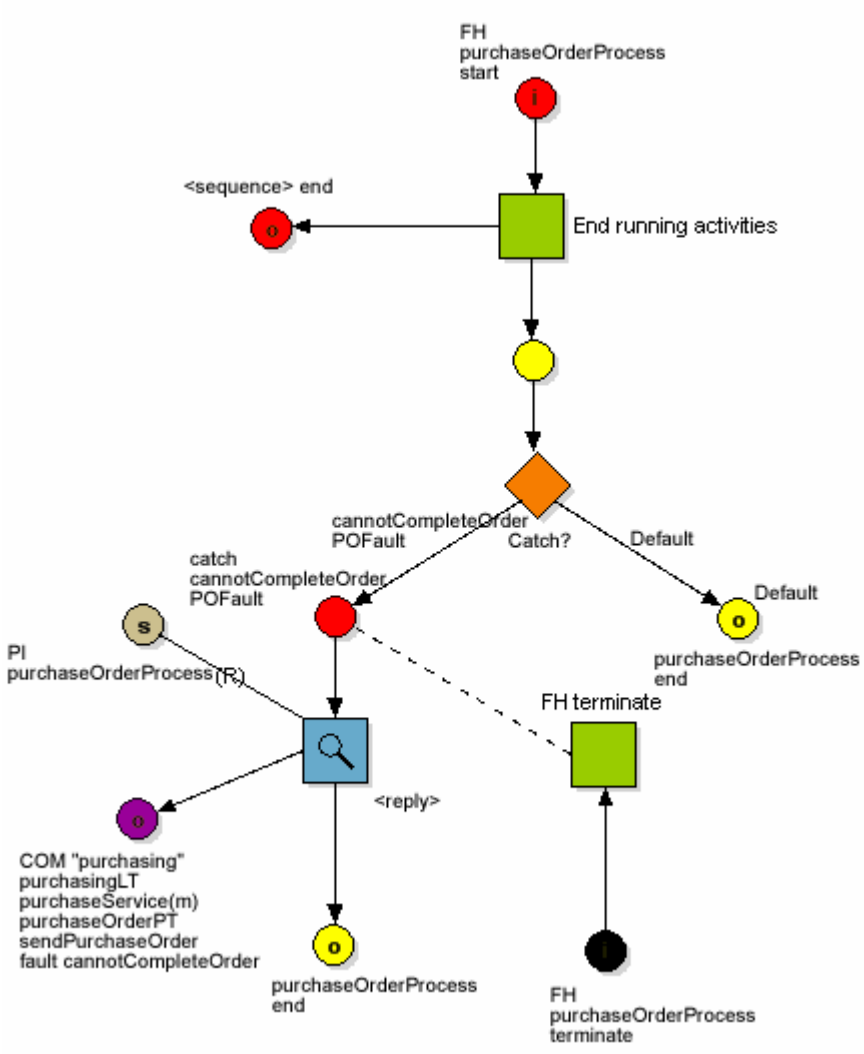


Figure G.3: Zooming in on the "FH purchaseOrderProcess" element of figure G.2.

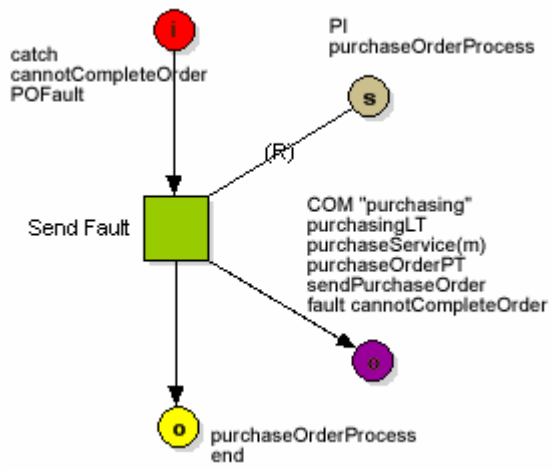


Figure G.4: Zooming in on the "<reply>" element of figure G.3.

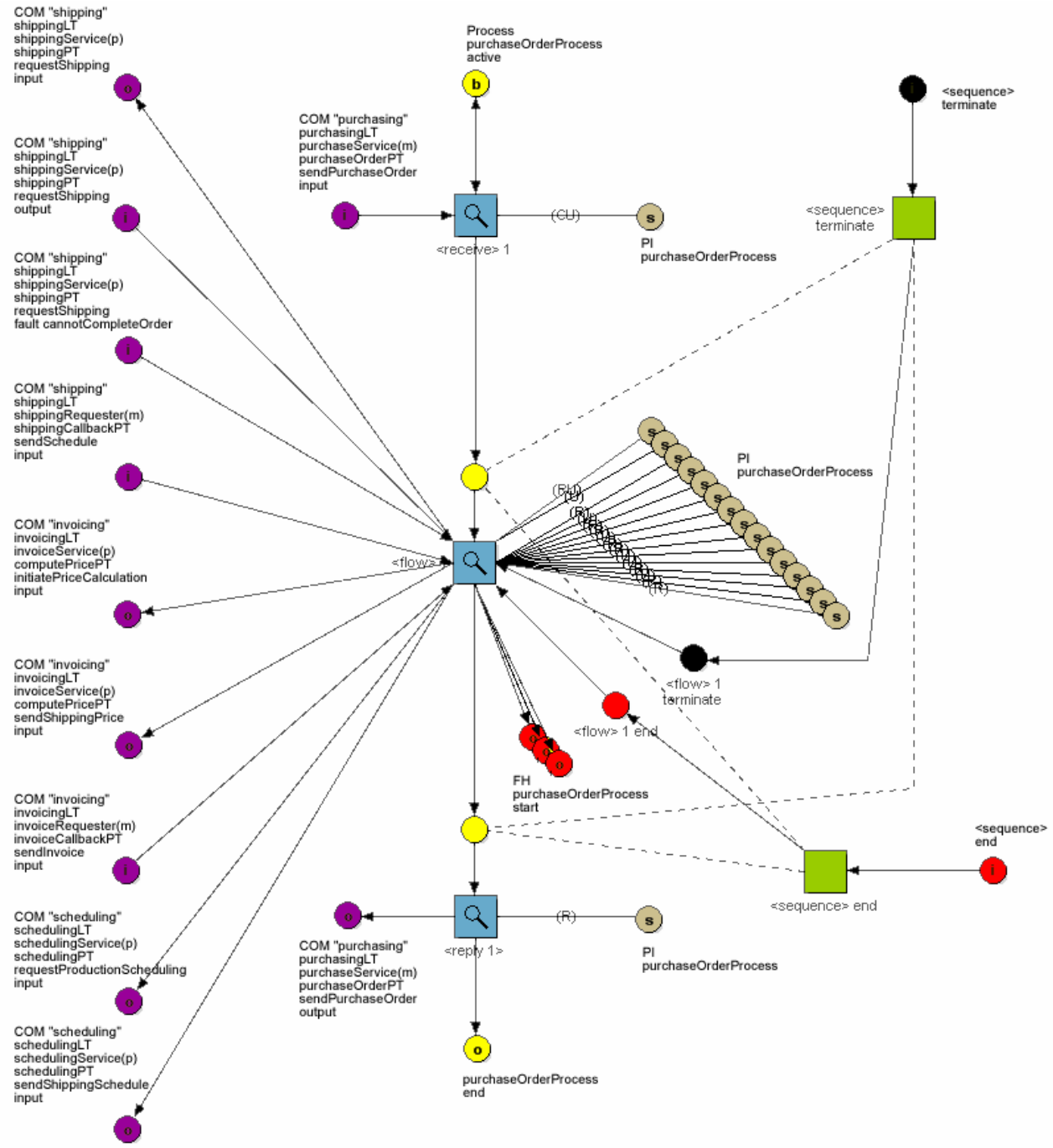


Figure G.5: Zooming in on the "<sequence>" element of figure G.2.

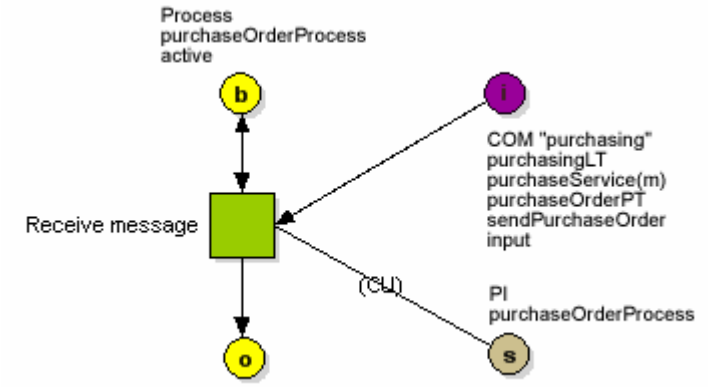


Figure G.6: Zooming in on the "<receive> 1" element of figure G.5.

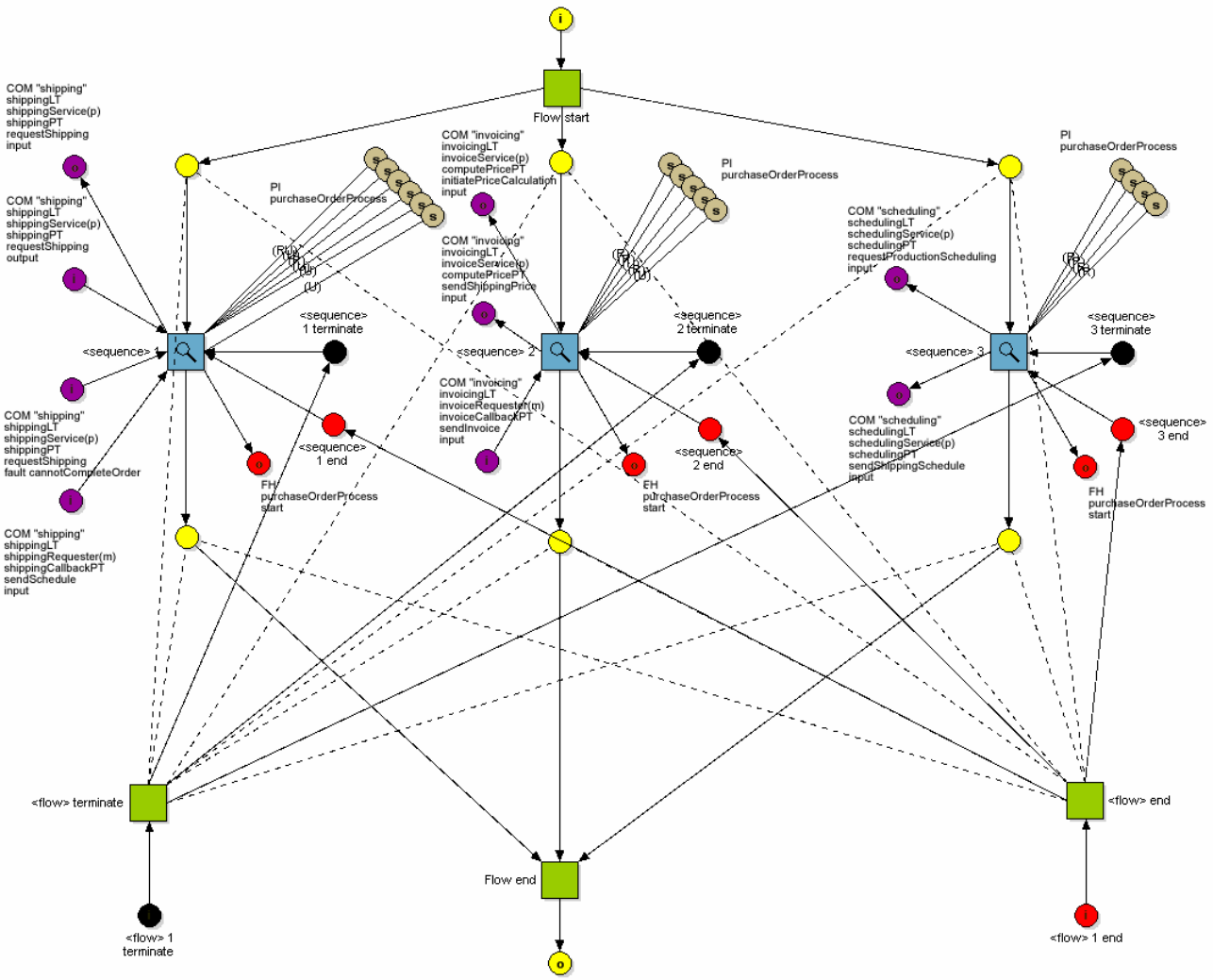


Figure G.7: Zooming in on the "<flow> 1" element of figure G.5.

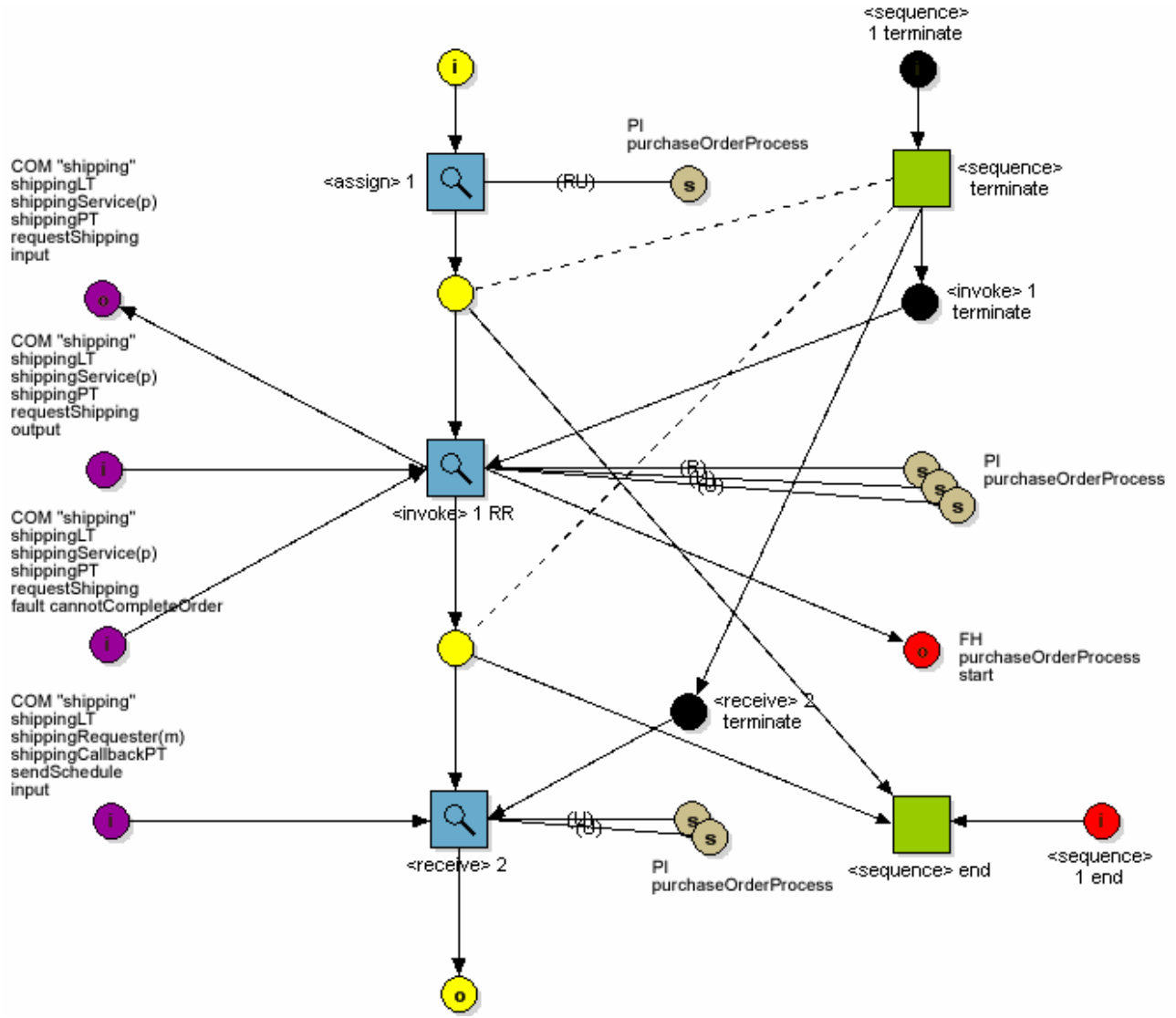


Figure G.8: Zooming in on the "<sequence> 1" element of figure G.7.

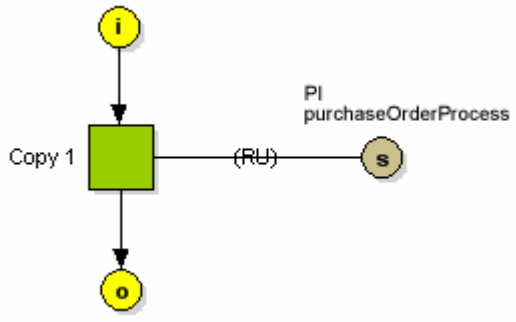


Figure G.9: Zooming in on the "<assign> 1" element of figure G.8.

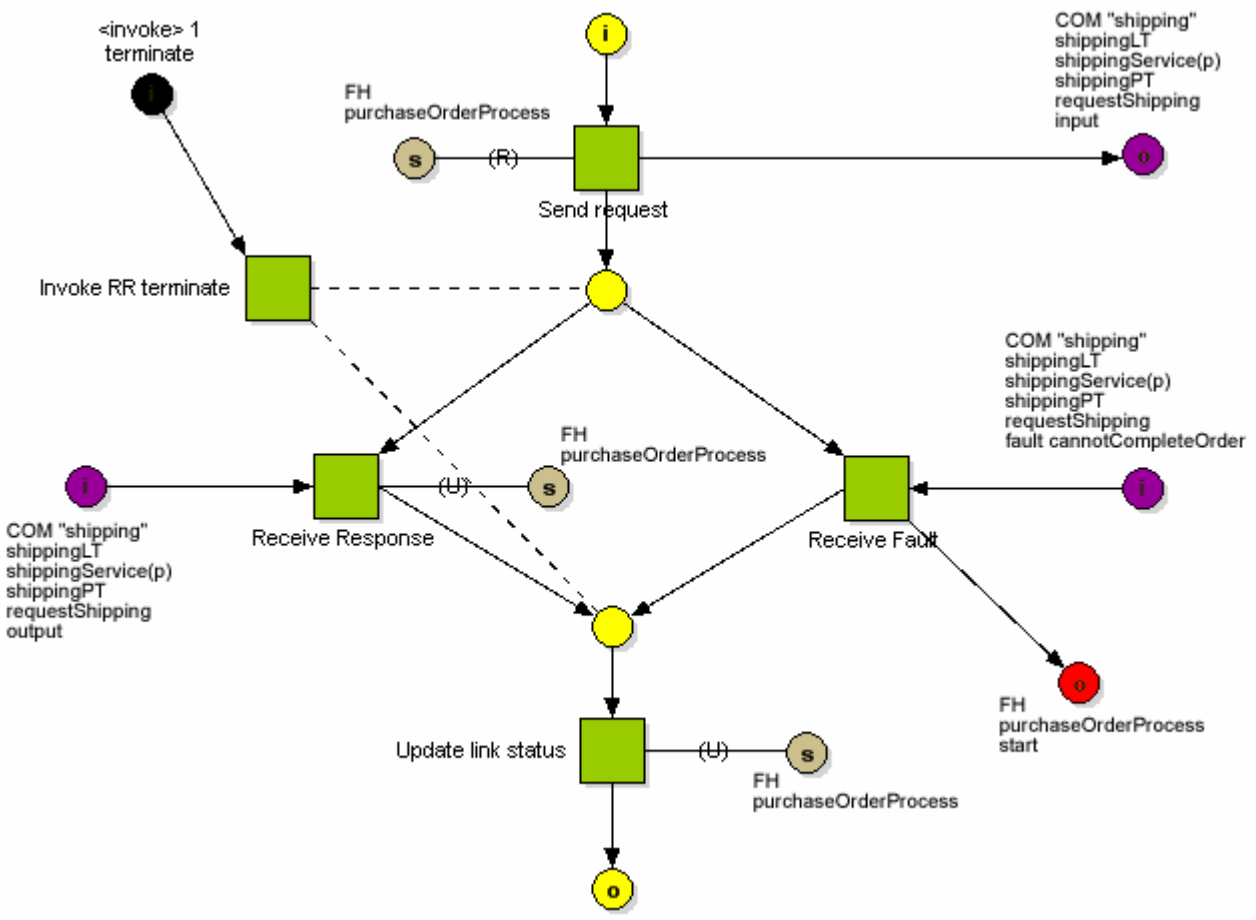


Figure G.10: Zooming in on the "<invoke> 1 RR" element of figure G.8.

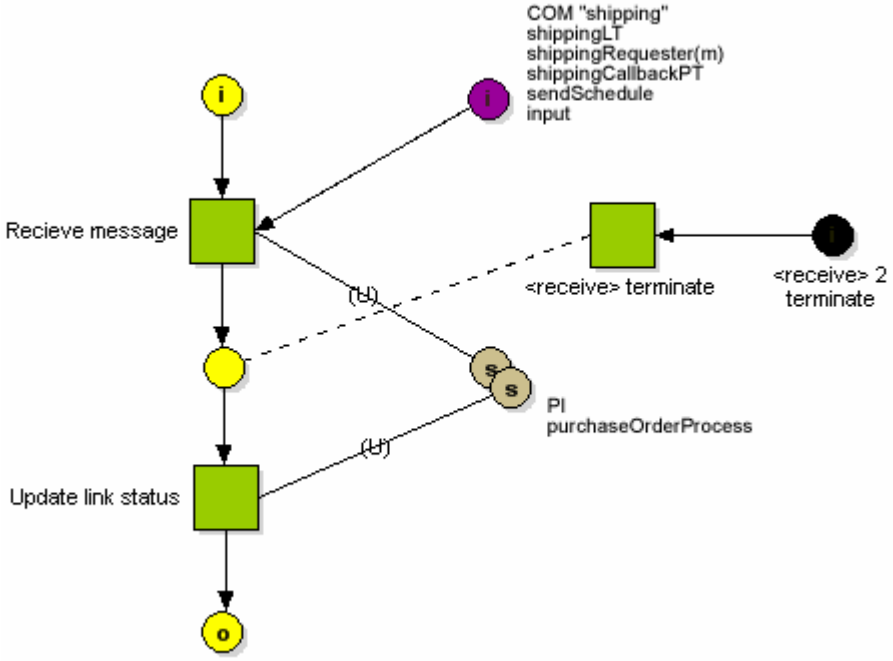


Figure G.11: Zooming in on the "<receive> 2" element of figure G.8.

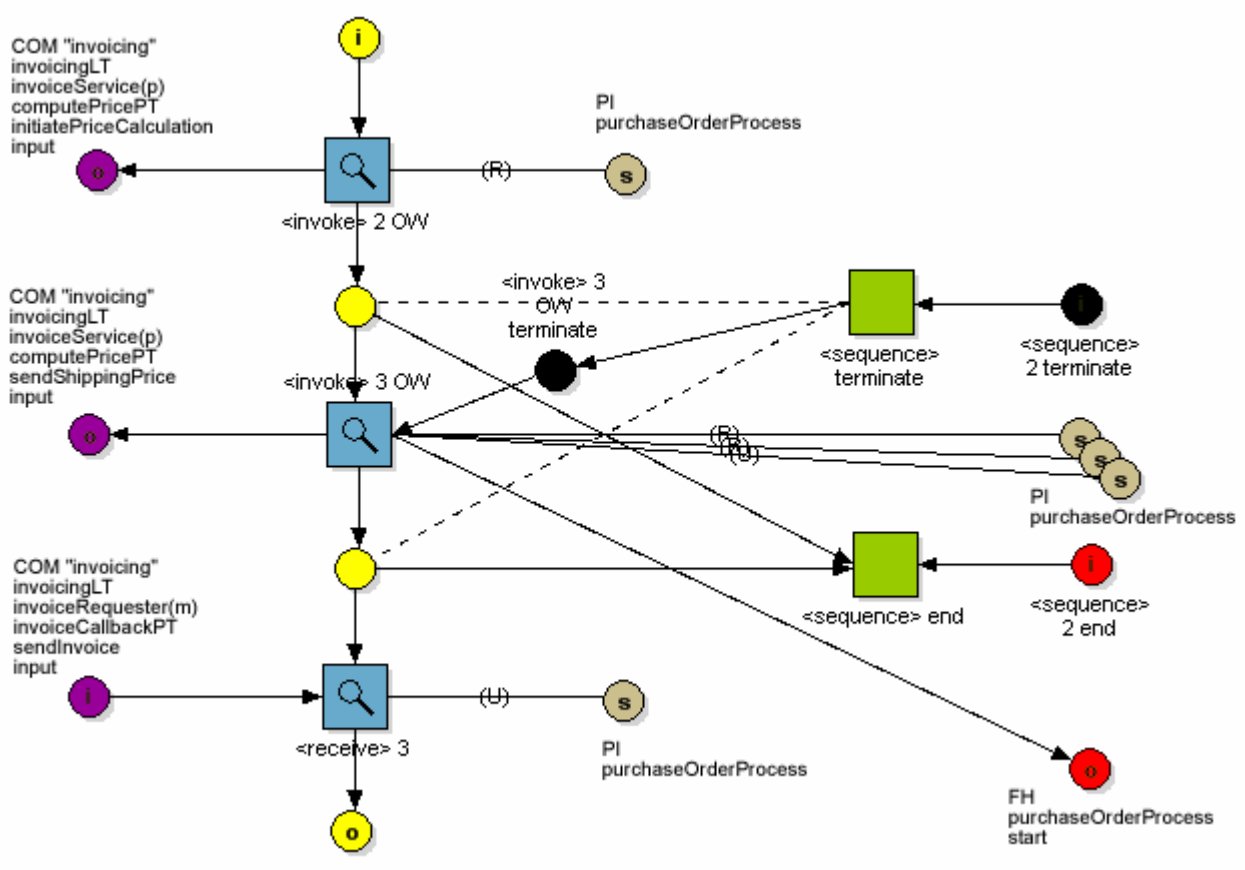


Figure G.12: Zooming in on the "<sequence> 2" element of figure G.7.

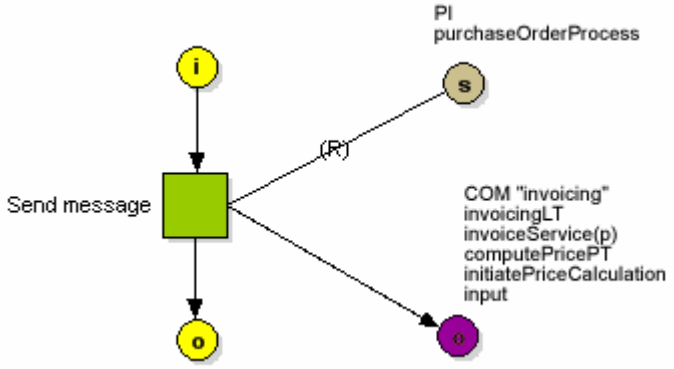


Figure G.13: Zooming in on the "<invoke> 2 OW" element of figure G.12.

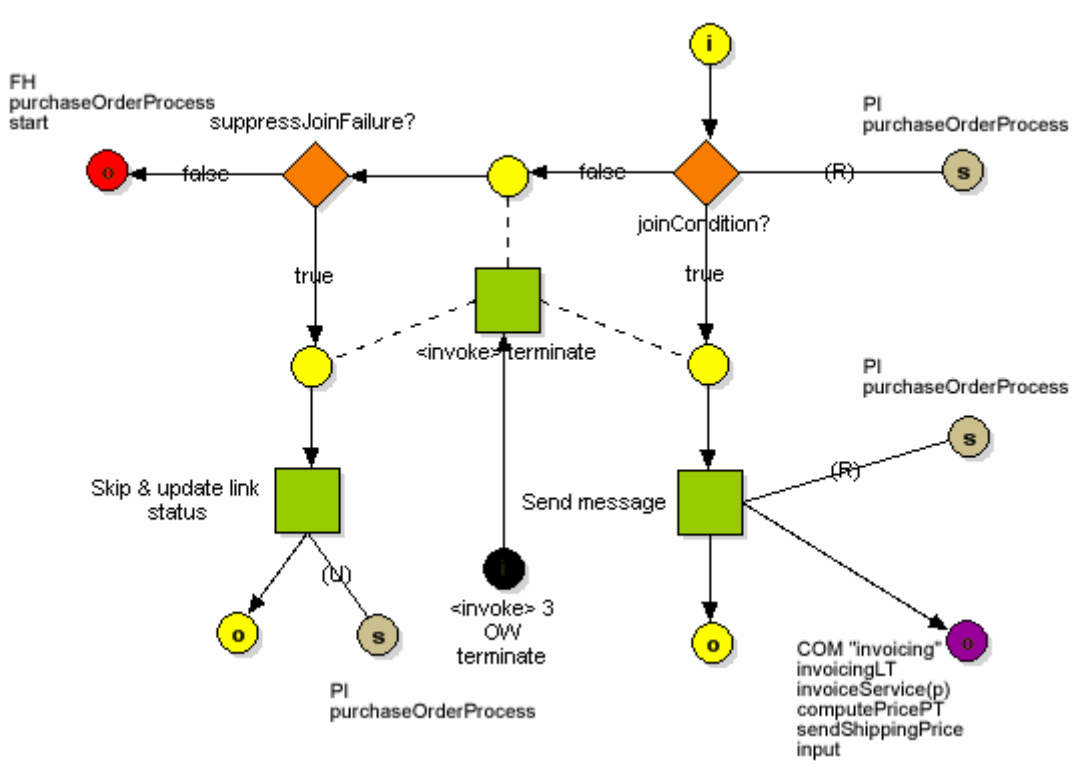


Figure G.14: Zooming in on the "<invoke> 3 OW" element of figure G.12.

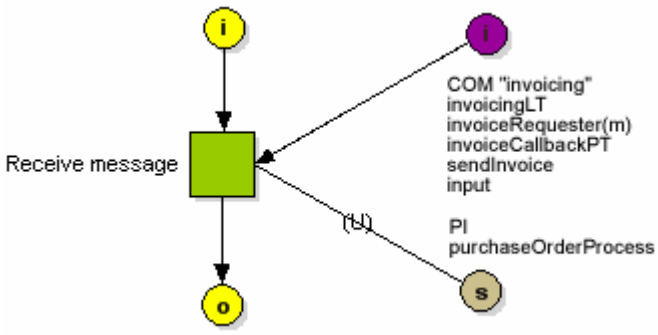


Figure G.15: Zooming in on the "<receive>3" element of figure G.12.

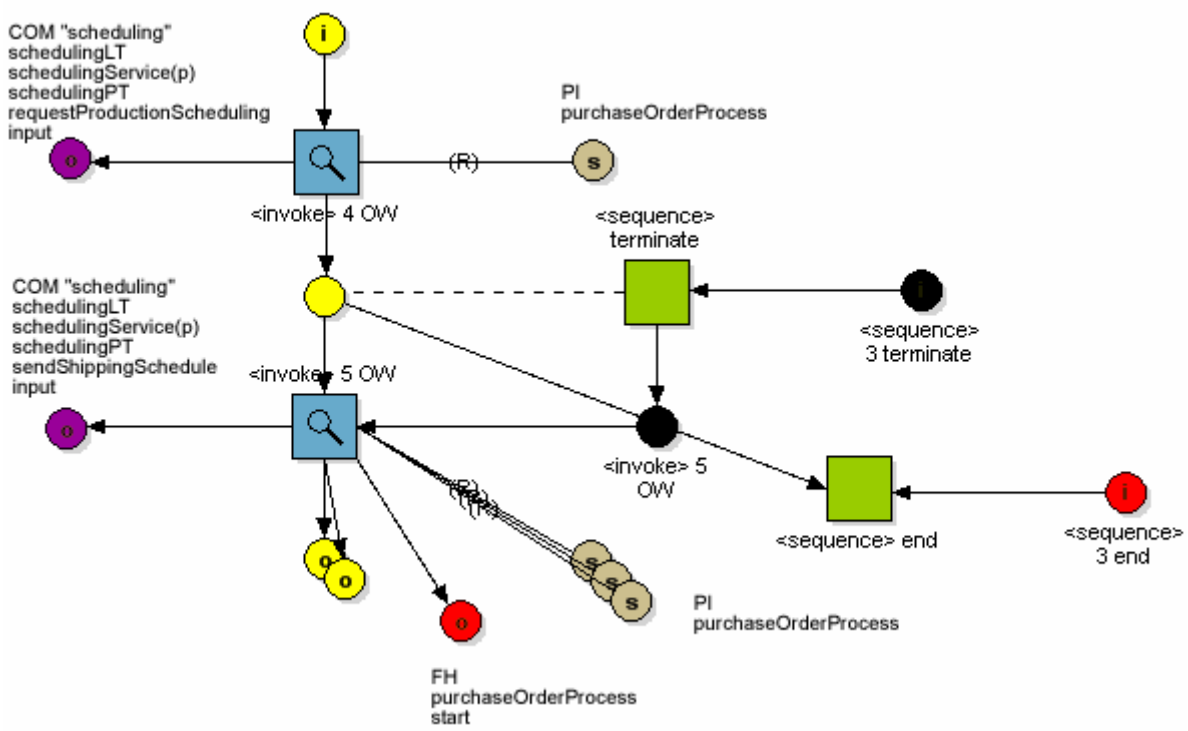


Figure G.16: Zooming in on the "<sequence> 3" element of figure G.7.

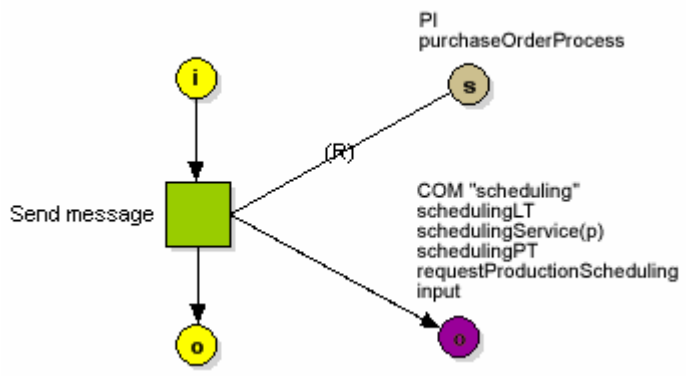


Figure G.17: Zooming in on the "<invoke> 4 OW" element of figure G.16.

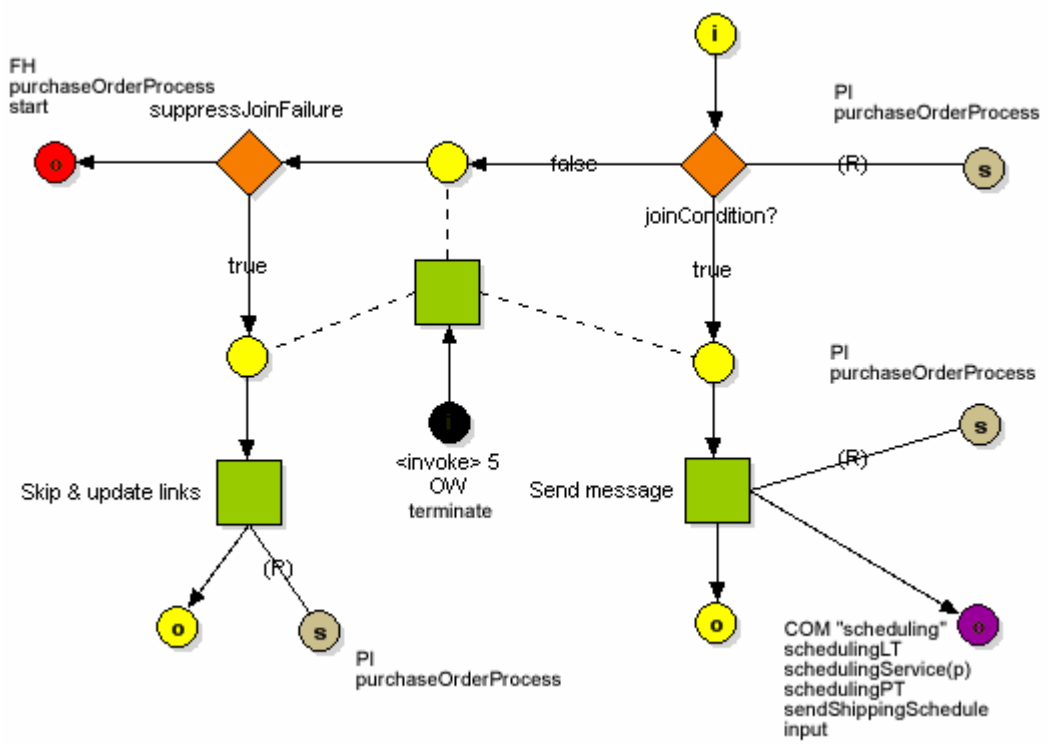


Figure G.18: Zooming in on the "<invoke> 5 OW" element of figure G.16.

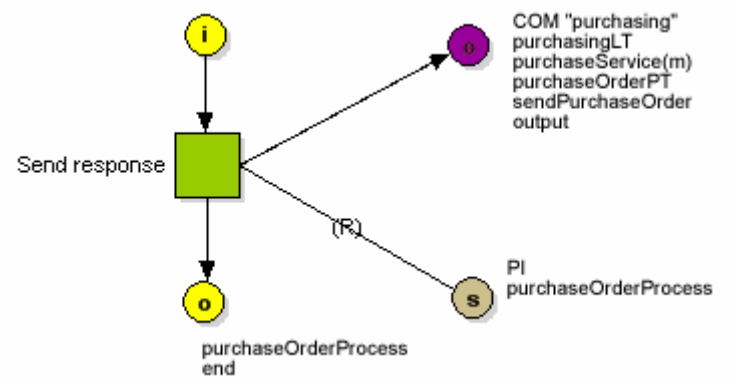


Figure G.19: Zooming in on the "<reply> 1" element of figure G.5.

References

[BPEL4WS, 2003] "Business Process Execution Language for Web Services, version 1.1",
Authors: Tony Andrews (Microsoft), Francisco Curbera (IBM), Hitesh Dholakia (Siebel Systems), Yaron Goland (BEA), Johannes Klein (Microsoft), Frank Leymann (IBM), Kevin Liu (SAP), Dieter Roller (IBM), Doug Smith (Siebel Systems), Satish Thatte (Microsoft, editor), Ivana Trickovic (SAP), and Sanjiva Weerawarana (IBM),
Companies: BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems,
5 May 2003,
URL:

- BEA: http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp
- IBM: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
- Microsoft: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/bpel1-1.asp
- SAP: http://ifr.sap.com/bpel4ws
- Siebel: http://www.siebel.com/bpel

[BPMN, 2004] "Business Process Modeling Notation (BPMN), version 1.0",
Authors: Stephen A. White and members of the BPMI Notation Working Group,
Institute: BPMI.org,
3 May 2004,
URL: http://www.bpmi.org/BPMN.htm

[PN, 1962] "Kommunikation mit Automaten",
Author: C.A. Petri,
Institute: Bonn, Institut für Instrumentelle Mathematik,
1962

[CSS2, 1998] "Cascading Style Sheets, level 2; CSS2 Specification",
Editors: Bert Bos, Håkon Wium Lie, Chris Lilley and Ian Jacobs,
Institute: W3C
12 May 1998,
URL: http://www.w3.org/TR/REC-CSS2/

[EPNML, 2004] "EPNML 1.1 – an XML format for Petri nets",
Authors: J.M.E.M. van der Werf and R.D.J. Post,
Institute: Eindhoven University of Technology,
14 June 2004,
URL: http://www.petriweb.org/specs/epnml-1.1/pnmldef.pdf

[HLPN, 2000] "High-level Petri Nets – Concepts, Definitions and Graphical Notation", Final Draft International Standard ISO/IEC 15909, Version 4.7.1, 28 October 2000, URL: http://www.petrinets.info/docs/pnstd-4.7.1.pdf

[IBM BPEL 1, 2002] "Business Process with BPEL4WS: Understanding BPEL4WS, Part 1" Authors: Sanjiva Weerawarana, Francisco Curbera, Company: International Business Machines Corporation, August 2002, URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/

[IBM BPEL 2, 2002] "Business Process with BPEL4WS: Learning BPEL4WS, Part 2" Author: Rania Khalaf, Company: International Business Machines Corporation, August 2002, URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol2/

[IBM BPEL 3, 2002] "Business Process with BPEL4WS: Learning BPEL4WS, Part 3" Authors: Matthew Duftler, Rania Khalaf, Company: International Business Machines Corporation, October 2002, URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol3.html

[IBM BPEL 4, 2002] "Business Process with BPEL4WS: Learning BPEL4WS, Part 4" Author: Nirmal Mukhi, Company: International Business Machines Corporation, November 2002, URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol4/

[IBM BPEL 5, 2003] "Business Process with BPEL4WS: Learning BPEL4WS, Part 5" Authors: Matthew Duftler, Francisco Curbera, Rania Khalaf, Company: International Business Machines Corporation, March 2003, URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol5/

[IBM BPEL 6, 2003] "Business Process with BPEL4WS: Learning BPEL4WS, Part 6" Authors: Rania Khalaf, William A. Nagy, Company: International Business Machines Corporation, March 2003, URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol6/

[IBM BPEL 7, 2003] "Business Process with BPEL4WS: Learning BPEL4WS, Part 7" Authors: Rania Khalaf, William A. Nagy, Company: International Business Machines Corporation, April 2003, URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol7/

[IBM BPEL 8, 2003] "Business Process with BPEL4WS: Learning BPEL4WS, Part 8"
Authors: Rania Khalaf, Nirmal Mukhi,
Company: International Business Machines Corporation,
May 2003,
URL: http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol8/

[IBM WS I, 2002] "Transactions in the world of Web Services, Part 1",
Authors: Tom Freund and Tony Story,
Company: International Business Machines Corporation,
August 2002,
URL: http://www-128.ibm.com/developerworks/webservices/library/ws-wstx1/

[ORACLE, 2004] "Quick Start Tutorial for the Oracle BPEL Process Manager",
Company: Oracle,
2004,
URL: http://download.oracle.com/otndocs/products/bpel/quickstart.pdf

[PNML, 2003] "The Petri Net Markup Language: Concepts, Technology, and Tools"
Authors: Jonathan Billington (University of South Australia), Søren Christensen (University of Aarhus), Kees van Hee (Eindhoven University of Technology), Ekkart Kindler (University of Paderborn), Olaf Kummer (Core Media), Laure Petrucci (Laboratoire Spécification et Vérification), Reinier Post (Eindhoven University of Technology), Christian Stehno (Carl von Ossietzky University Oldenburg), and Michael Weber (Humboldt-Universität zu Berlin),
Institutes: University of South Australia, University of Aarhus, Eindhoven University of Technology, University of Paderborn, Core Media, Laboratoire Spécification et Vérification, Carl von Ossietzky University Oldenburg, and Humboldt-Universität zu Berlin,
March 2003,
URL: http://www.informatik.hu-berlin.de/top/pnml/download/about/PNML_CTT.pdf

[ResetPN, 1989] "Extensions to Coloured Petri Nets and their Application to Protocols",
Author: J. Billington,
Institute: University of Cambridge,
1989

[TPN, 1998] "Timed Petri Nets, Theory and Application",
Author: Jiacun Wang,
Institute: Monmouth University,
1998,
URL: http://bluehawk.monmouth.edu/~jwang/TPN.htm

[UML, 2005] "Unified Modeling Language: Superstructure, version 1.4.2",
Institute: Object Management Group (OMG),
January 2005,
URL: http://www.omg.org/cgi-bin/doc?formal/04-04-04

[WBERG, 2003] "Grafische Repräsentation von BPEL4WS",
Author: Daniela Weinberg,
Institute: Humboldt-Universität zu Berlin,
15 August 2003,
URL: http://www.informatik.hu-berlin.de/top/download/publications/weinberg03.pdf

[WfMC, 1999] "Workflow Management Coalition Terminology & Glossary", Issue 3.0"
Institute: Workflow Management Coalition,
February 1999,
URL: http://www.wmfc.org/

[WSDL, 2001] "Web Services Description Language (WSDL) 1.1",
Authors: Eric Christensen (Microsoft), Francisco Curbera (IBM), Greg Meredith (Microsoft), and Sanjiva Weereawarana (IBM),
Companies: Ariba, International Business Machines Corporation, Microsoft Corporation,
15 March 2001 (W3C Note),
URL: http://www.w3.org/TR/wsdl

[WSFL, 2001] "Web Services Flow Language (WSFL), version 1.0",
Author: Prof. Dr. Frank Leymann,
Company: International Business Machines Corporation,
May 2001,
URL: http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[XLANG, 2001] "XLANG, Web Services for Business Process Design",
Author: Satish Thatte,
Company: Microsoft Corporation,
2001,
URL: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[XSD, 2004] "XML Schema Part 2: Datatypes Second Edition",
Authors: Paul V. Biron and Ashok Malhotra,
Companies: Kaiser Permanente and Microsoft Corporation,
W3C Recommendation 28 October 2004,
URL: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html

[YASPER, 2005] "YASPER User Guide",
Author: R.Post,
Institutes: Eindhoven University of Technology and Deloitte,
13 May 2005,
URL: http://www.yasper.org

[Z-NOT, 1998] "The Z-Notation: A Reference Manual, second edition",
Author: J.M. Spivey,
Institute: University of Oxford,
1998,
URL: http://spivey.oriel.ox.ac.uk/~mike/zrm/zrm.pdf